# Simulation of Corticogenesis as a Self-Organizing System

Dissertation zur

Erlangung der naturwissenschaftlichen Doktorwürde

(Dr. sc. nat.)

vorgelegt der

Mathematisch-naturwissentschaftlichen Fakultät

der

Universität Zürich

von

Frédéric Zubler

Hunzenschwil (AG)

Promotionskomitte

Prof. Rodney J. Douglas

Prof. Kevan A. C. Martin

Zürich, 2009

# Acknowledgements

During my stay at the Institute of Neuroinformatics, I had the chance to be surrounded by brilliant, helpful and friendly people. I would like to thank them for their help over the last five years; in particular:

- Prof. Rodney Douglas, for his support, guidance and teaching, and for having given me the opportunity to join the INI and work on this project.

- Prof. Kevan Martin for his benevolent influence.

- Prof. Esther Stoeckli, for accepting to be on my examination committee.

- Fabian Roth who guided my first steps, Dennis Goehlsdorf who literally saved my life with his triangulation, Andreas Hauri who's programming talent strengthened the project, Sabina Pfister for sharing her biological models, Roman for his mathematical support, Matthew Cook who's ease at solving any king of riddle has changed my way to consider problems in general, and all my colleagues and friends from the R&F's meetings, Daniel Rubin, Joerg Conrad, Andreas Steimer, Bobby Rohrkemper, Dylan Muir, Jason Rolfe, Emre Neftci, Christoph Krauz, Florian Jug, Tobby Weston, Klaus Hepp and Adrian Whatley. For the friendly chats and the hard work sessions. For the beers and the heissi shoggi.

- All my INI friends, who taught me to play chess or the guitar, helped to improve my english/german/swiss german/japanese, who took me to lunch with them or showed me neurons on the microscope: Raphael, Jerome, Niels, Steven, Vasco, Tom, Yingxue, Isabelle, Alumit, John, Nuno, Georg, Markus, MC, Paul, Cyrille, and many others.

- The Admin-team.

Thank you to my family, and in particular to Nathalie, for her continuous support.

# Summary

During the past two decades, the focus of biological investigation has begun to shift from reductive analysis of low level mechanisms toward an understanding of how these mechanisms interact in complex ways to elicit large scale organized processes. This shift is clearly seen in neuroscience and molecular biology where there is a thrust to understand the behavioral expression of neuronal interactions, or the control of biochemical pathways in terms of molecular networks. Simulation is a useful tool for exploring such complex processes because it permits rigorous analysis of observed global behavior in terms of the mechanistic axioms declared in the simulated model.

In the first part of this thesis (Chapter 2 and 3), we present CX3D, a software package for simulation of neural growth and network development in a physically realistic 3D environment. Chapter 2 describes the physical properties of tissue in our simulator: Neurons are discretized into spheres (for the soma) and chains of cylinders (for the neurites). Three types of mechanical interactions are applied to these discrete neuronal elements: inter-object forces when two elements are in close contact, intra-object forces if neurites are stretched and biological movements (cell movements). These three forces are summed up, and each object moves accordingly. To maintain a neighborhood relation between the objects in the simulator, we use a dynamic Delaunay triangulation. This triangulation (or more exactly its dual graph) provides us with a decomposition of the extracellular space, which we use for the simulation of diffusing of signaling molecules.

Chapter 3 describes the software architecture of CX3D, and its four levels of abstraction: the Delaunay triangulation, the physical level (mechanical and diffusive properties) and two levels that are used by the modeler to code the specificities of the cells. We describe how the biological properties of neurons in our simulator are encapsulated into small modules linked to particular physical objects. A commented example guides the reader through the re-implementation, in our simulator, of a famous model of axonal branching proposed by van Ooyen and collaborators. We also present several other models of neural

development, illustrating the versatility of our simulator.

The second part of the thesis (Chapter 3 and 4) introduces a language for the control and explicit programming of self-assembling of cortical circuits. In Chapter 3, we formalize the description of neural development by defining a set of primitive actions taken locally by neural precursors during corticogenesis. These primitives can be combined into networks of instructions, similar to biochemical pathways, capable of reproducing arbitrarily complex developmental sequences in a biologically plausible way. Furthermore, the conditional activation and deactivation of these instruction networks can also be controlled by the usage of these primitives, allowing for the design of a 'genetic code', with coding elements and regulating elements. We show in simulation how such a code can be incorporated into a single initial progenitor, that then reproduces the major steps of corticogenesis, developing into a multilayer connected neural network.

In the final Chapter, we apply the same formalism to more detailed models of corticogenesis from mitotic division of progenitor cells, through formation of distinct populations of different cell types, migration of neuronal precursors to form the cortical laminae and finally extension of axons and dendrites reproducing the experimentally determined branching patterns of neurons in the cat visual cortex.

# Zusammenfassung

Zunehmend richtet sich in der Biologie der Focus von der Analyse der Basisprozesse hin zum Verstehen der Integration der Mechanismen in höheren Organisationsstufen. Dieser Wechsel ist klar sichtbar in Neurowissenschaften und Molekularbiologie; das Ziel ist, zu verstehen wie biochemische Prozesse neuronale Netzwerke bilden, und diese dann Verhalten. Simulation ist ein wichtiges Instrument zur Untersuchung solch komplexer Prozesse: man kann exakt untersuchen wie globales System-Verhalten aus den mechanistischen Axiomen in einem Modell entsteht.

Im ersten Teil der Dissertation präsentieren wir CX3D, ein Programm zur Simulation von Netzwerkbildung aus Neuronenvermehrung in einer physikalisch realistischen 3D-Welt. Kapitel 2 beschreibt die physikalischen Eigenschaften von Zellgeweben in unserem Simulator. Die Neuronen sind repräsentiert als Objekte mit mehreren diskreten Elementen: eine Kugel (Zellkörper) verbunden mit Ketten von Zylindern (Axon und Dendriten). Drei Arten von Wirkungen beeinflussen die Objekte/Elemente: Mechanische Kräfte zwischen Objekten (Kollisonen zwischen Neuronen) und innerhalb Objekten (z.B. Zugkräfte im Axon), und Selbstbewegung von Zellen. Ihre Summe bestimmt die Objektbewegungen. Zur Bestimmung von Nachbarschafts-Beziehungen verwenden wir eine dynamische Delauny-Triangulation (die dual graph Methode). Dies erlaubt eine Raumaufteilung, die auch für die Simulation der Diffusion von Signalmolekülen wichtig ist.

Kapiel 3 beschreibt weitere Aspekte von CX3D; die vier Abstraktions-Ebenen: die Triangulation, die Mechanik/Diffusions- Vorgänge, und zwei Ebenen für die Kodierung spezifischer Zelleigenschaften. Wir erklären, wie die biologischen Eigenschaften in Module eingegeben werde, welche mit Objekten verbunden werden. Ein Beispiel führt den Leser durch unsere Simulation des berühmten Modells für Axon-Verzweigung von Van Ooyen u. Mitarbeitern. Noch andere Beispiele für Neuronen-Entwicklung zeigen die mannigfaltige Anwendbarkeit unseres Simulators.

Der zweite Teil stellt eine neue Programmierung zur Kontrolle von sich selbst bildenden Netzwerken

vor. Wir formalisieren dies mit der Definition einer Serie von Elementar-Funktionen der unreifen Zellen. Deren Kombination führt zu vernetzten Informationen ähnlich wie biochemische Prozesse und erlaubt für arbiträr komplexe Entwicklungen auf eine biologisch plausible Weise. Konditionelle Aktivierung/Desaktivierung von Funktionen wird ebenfalls durch Elementar-Funktionen kontrolliert wie durch einen genetischen Code, der auch die Codierung reguliert. Wir zeigen, wie eine einzige Zelle, mit eingebautem Instruktions-Code, wichtige Stadien der Formation des Kortex reproduzieren kann, mit Bildung eines mehrschichtigen Neuronen-Netzwerkes.

Im Schlusskapitel verwenden wir den Formalismus für mehr detaillierte Modelle für Kortikogenese durch mitotisch aktive Zellen: von der Bildung diskreter Populationen von different Zell-Typen, und der Formation von Kortex-Schichten durch Zellmigrationen, bis zum Auswachsen von Axonen und Dendriten, das die für die Seh-Rinde der Katze beschriebenen Verästelungen reproduziert.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Purpose of this thesis

Unlike human artifacts where all components are specified in a blueprint and then assembled by an external constructor, biological tissues arrange and configure themselves by replication and interaction between their elements. The efficiency of these principles are especially impressive in the development of the neocortex, where the generation of a precise cytoarchitecture is necessary to support the computational cabilities of the brain. Current efforts in developmental neuroscience research mainly focus on reductive characterization of specific biological processes. Spectacular progresses have been made in the study of biochemical pathways and gene expression patterns [1–3]. This line of work is indispensable to understand development. But even if we had a complete description of all existing biochemical reactions, the level of complexity is such, that it would be impossible for a human mind to fully grasp the properties of such networks. As we will see in this thesis, simulation is a useful tool for exploring these complex processes as it permits rigorous analysis of observed global behavior in terms of the mechanistic axioms declared in the simulated model [4]. Applied to neural growth, simulation offers a means to grasp how the various interactions between dividing cells lead relentlessly to global neural network organization.

Understanding the organizational key principles involved in brain development would be of the highest interest for biology and medicine, but also for engineering, where the complexity of hardware and software systems has become such, that a pure feed-forward approach is reaching its limits [5]. Since the beginning of human history, many inventions were influenced by the observation of natural phenomena, from flying machines to sophisticated genetic algorithms applied to electronic circuit programming. If

they were understood, the principles leading to the formation of such an effective and robust computational system such as the mammalian neocortex would certainly inspire the development of entire new fields of applications.

But if we want to apply the principles of self-organization learned from biology to technology, understanding already existing systems is not sufficient. We also need to be able take the problem from the other end: Knowing the desired end-product, we need to specify the low-level principles that will generate it, in a self organizing manner. As R. Nagpal [6] states it: "Rather than observing emergent global behavior from given local rules, how does one derive local rules for a particular global goal? What are the high-level languages for describing global goals, and what are the primitives for constructing local rules?". This work aims at answering this question in the context of neural networks.

## 1.2 Need for a new simulation framework

Currently, a neuroscientist who decides to simulate a particular neuro-developmental problem has no simulation platform at disposal. And before designing his growth model, he has to go through the time-consuming process of programming his own modeling environment. This situation contrasts with the numerous excellent free softwares available to simulate electrophysiology of neurons, like for instance NEURON [7] and GENESIS [8] for multi-compartments models, or PCSIM [9] for point neurons models. By providing the scientists with the building blocks and the environment in which to assemble their own models, these programs have allowed users to design sophisticated simulations [10]. Similarly, researchers in other areas of biology also have at their disposal powerful modeling environment, for instance for the study of biochemical pathways [11].

In addition to allowing for simulations of various aspects of development (cell division, migration, extension of neurite and finally establishment of synaptic connections), a general purpose simulation platform should also emulate the physics of developing tissues, namely mechanics and diffusion. Mechanical forces influence both structural properties (such as cell densities and macroscopic architecture [12]) and functional properties (via influences on intracellular biochemical pathways, a mechanism called mechanotransduction [13]) of developing tissues. For instance in neurons, the tension in a neurite influences its shape [14], and its growth rate [15], and can determine between an axonal vs. dendritic fate [16]. In addition to mechanics, the development of biological tissues depends strongly on the ability of cells to communicate and influence one another, either by contact [17] or by release of diffusible signaling molecules [18].

Before being able to perform neural growth simulations, we had first to develop a simulation environment. The simulation package CX3D, the description of which forms the first part of this thesis, is an open-source software written in Java for modeling all stages of corticogenesis. It provides a simulated physical space governed by a physics engine which computes the forces between objects, and the diffusion of substances through the space. To define the desired cellular functionality, modelers write the code for small biological modules. These modules are then inserted into the cells or cell components of the simulation, and so express their respective specific biological properties. These modules could for instance describe the local environmental conditions for movement, neurite extension, and secretion of diffusible substances. The physics engine of CX3D then manages actual displacement, possible collisions, and simulates the diffusion of the secreted substances, whose concentration and gradient can be probed by any object in the simulation. In addition, our software allows the export of the grown architecture to an electrophysiology simulator. The source code and a user tutorial are freely available at http://www.ini.uzh.ch/projects/cx3d/.

## 1.3 An instruction language for the explicit programming of self-assembling cortical circuits

It is still unclear how the instructions leading to the formation of the neocortex are implicitly coded in the genome, and how they are expressed sequentially by the interactions among differentially activated genes. We study this problem in simulation in the second part of this work. For this purpose, we formalize the description of neural development by defining a set of primitive actions taken locally by neural precursors during corticogenesis. These primitives can be combined into networks of instructions, similar to biochemical pathways, capable of reproducing arbitrarily complex developmental sequences in a biologically plausible way. Moreover, the conditional activation and deactivation of these instruction networks can also be controlled by the usage of these primitives, allowing for the design of a 'genetic code', with coding elements and regulating elements. We show in simulation how such a code can be incorporated into a single initial progenitor, that then reproduces the major steps of corticogenesis, reproducing the experimentally determined morphology of several cell types of the cat visual cortex.

## 1.4 State of the art in neural growth simulation

Previous works on simulating neural growth can probably be classified in one of the following three categories, depending on the authors motivations: (1) Graph generation, developed to provide artificial neural network topologies, (2) Construction algorithms, used for generating structures reproducing the geometric characteristics of adult cells, and (3) Biologically-plausible growth models, for studying mechanisms underlying development.

### 1.4.1 Artificial neurons and genetic algorithms

In Artificial Neural Network (ANNs), a network of elementary units ("neurons") perform simultaneously rather simple local integrations while the collective behavior can represent a complex non-linear mapping. For a given architecture (i.e. a given directed graph whose vertices are the cells and the edges represent the possible synapses), many learning rules are known to adjust optimally the weights of the connections [19]. But no similar algorithms have been proposed to choose a right architecture, or to make it evolve. Therefore engineers or computer scientists interested in having a specific problem solved have tried to implement growth rules to create a network. To maximize the performance of the network for a particular task, genetical algorithms (GA) [20, 21] can be used to evolve the growth rules[1]. For some authors there is little concern to stay as close as possible to biology, but rather to get inspired by some aspects of it to solve a particular problem. Other groups have been more inspired by biology to implement their growth procedures.

The first works implied a direct coding scheme (direct here means that there is a direct mapping of the genotype onto the phenotype), like for instance a chromosome describing the neural connectivity matrix (reviewed in [23]). In addition from being totally biologically and physically un-plausible, this explicit encoding has the drawback that the genetic code increases with a quadratic relation to the size of the network. Grammar-based models like the L-systems described previously are an elegant way to circumvent the latter point, in which very simple rules can give rise to extremely complicated networks [24–26]. Nevertheless they are as far from the biological processes than direct encoding [27]. And by being usually context-insensitive, they don't take into account the numerous cell-cell interactions, which play an essential role during development. More biologically plausible mechanisms were developed for the establishment of connections between already pre-defined neuron somata, for instance

[1] Some authors actually used GA alone to define the actual weights, instead of defining first the topology and then training the weights [22]

by modeling gradient-based axon guidance [28]. In addition to solely describing the connections, several groups included to the genetic code instructions for cell migration and division, making a step further toward growth of a neural population [29, 30]. Eggenberger [31] first added a gene regulatory network to neural growth simulator, meaning he considered two types of genes: The first time of genes code for transcription factors, that will regulate the activity of the second type of genes, that coding for a function in development (like the expression of a membrane receptor).

Most of the authors cited above used evolutionary algorithms to design their genome in order to obtain a suitable circuit. This approach is fine if one is interested in getting the best network for a particular problem. It is still successfully used nowadays for solving practical problems in engineering, chemistry or medicine. Nevertheless it doesn't help us to understand the principles relating functionally a gene network to a neural network [5]. There were very few attempts to explicitly *program* a genetic code to grow desired networks. Gruau et al. [32] published a compiler that would take a program written in Pascal as input, and generate (in a biologically un-plausible way) a neural network performing the same operations. Roth et al. [33] proposed a more biologically realistic model of development for a multicellular organism with a simple neural system, capable of performing a foraging task. Interestingly, they explicitly designed the genetic code to generate the desired organism in a self-organizing manner. A further step was taken by Nagpal [6], although not for neural networks, who proposed an instruction language based on primitives, representing both a genome element and the action of a cell. She applied her framework to control the folding of an already-formed single layer of cells.

### 1.4.2 Construction algorithms in computational neuroanatomy

The construction approach aims at reproducing the geometrical properties of real neurons, and not at understanding the biological processes underlying neural growth. Often the motivation is to produce realistic dendritic structures to be used in an electrophysiology simulator (e.g [34]). The prototypical example of the construction approach is L-Systems, invented by Lindenmayer [35] to model plant morphology - and then quickly applied also to neurite morphology [36]. L-Systems use a string rewriting process, begining with an initial string of characters (the axiom) that is operated on by a set of production rules that are applied at each time step. The characters of the generated string have a 2D or 3D graphical interpretation that represents the morphology of the grown neurite. The recursive nature of L-Systems allows them to reproduce elegantly some fractal properties of biological neurons. Later, L-Systems were extended in order to generate more realistic trees by incorporating specific parameters

extracted from the statistical analysis of real neuronal morphology [37, 38]. These newer models come closer to genuine growth algorithms, like Galton-Watson branching processes [39], which simulate the extension of a tree by adding segments. At each time step the terminal branches have three possibilities: to elongate, to bifurcate, or to stop. In some models of branching processes, the branching probabilities is dependent on the position of each segment in the tree [40]. Many other methods have been proposed to generate neuronal structures, including Markov models [41], Monte Carlo processes [42], and more 'abstract' approaches like to model elongation by diffusion and adhesion of virtual particles [43]. But as successful as they are in reproducing neuronal shapes, these models provide very little insight to the fundamental growth mechanisms leading to cortex formation.

### 1.4.3 Developmental models

Growth models, on the other hand, study the biological mechanisms that underly the generation of neuronal morphology. Many interesting agent-based simulations have been published, reproducing various aspects of development, such as cell proliferation [44, 45], polarization [46], cell migration [47], neurite extension [48], growth cone steering [49–51], fasciculation [52] and synapse formation [53, 54]. Mean field models have also been proposed, for instance to study axon guidance and map formation [55, 56]. Unfortunately, all these studies were conducted within different frameworks, which prevents the comparison between models, or the further extension of the combination of several computational models in larger simulations.

## 1.5 Review on corticogenesis

Since this thesis focuses on the self-organizational principles in the context of cortical development, we summarize in this section some key steps of corticogenesis.

Neurulation - the formation of the neural tube - starts at a stage when the embryo consists of three strata : endoderm, mesoderm, ectoderm. Some cells from the ectoderm elongate into columnar cells (in response to signals produced by the mesoderm), bend and then invaginate into the mesoderm to form a hollow lumen, that separates from the surface ectoderm and will eventually become the central nervous system. This initially straight tube undergoes expansion of its cavity in the anterior part, that will form the ventricles. At the same time, the cells differentiate according to their position. The dorso-ventra specification is under the control of a series of genes including the Hox genes, and along

the anterior-posterior axis the differentiation is due to the concentration gradient of two substances: the Sonic Hedgehog protein produced ventrally by the notochord and the TGF-beta family produced dorsally [57–59].

During neurulation, neuroepithelium forming the neural plate folds into a tubular structure, the neural tube that will form the spinal cord, the brain stem and the brain. At that stage, the most rostral part that will become the telencephalon is composed of a single layer of epithelial proliferative cells, called the Vectricular Zone (VZ) [60]. This proliferation by symmetrical divisions thickens the epithelium and enlarges the area [61]. The neurogenesis starts when the cells loose some of their epithelial characteristics [62], and switch to an asymmetrical mode of division [61]. Several new types of cells appear in VZ, like the Radial Glial Cells (RGC) [63], which are attached to both the pial and the ventricular basal membrane and will be used by neuron precursors for their migration. Short neural precursors are another type of cells, that have lost the contact with the pial surface [64]. A third type of cells [65] divide near the apical surface and form a second germinative compartment, the sub ventricular zone (SVZ) [60].

Neuron precursors in the VZ and the SVZ express different markers, and are controled by different transciption factors. They also have a different faith: the VZ will form the lower layer neurons, whereas precursors from the SVZ will form the more superficial layers, which could have play a role in the specific expansion of the supragralular layers in the primate cortex compared to the rodent cortex [66].

The first neuronal cells that appear form the preplate (PP), which is composed of the Cajal-Rezius cells in the marginal zone (MZ), and of the subplate (SP) cells. The former will form the future layer 1, while the latter ones are located in the white matter and below Layer 6. The origin of these cells are not absolutely clear, although there is evidence that Cajal-Retzius cells might migrate tangentially from the caudomedial wall of the tellencephalic vesicles [67]. It seems that these layers are formed in an outside-in (older superficial to younger deep) [68], in contrast to the formation of the cortical plate which is inside-out.

At this stage, more and more cells exit the cell cycle and migrate radially from the VZ through the SP and stop before entering the MZ. As more of them arrive, they form a clearly visible layer, the cortical plate (CP) which pushes the MZ further up, splitting it from the SP. As development progresses, different cell types are produced sequentially, that migrate through their predecessors and form the different layers of the cortex in an inside-out manner: first layer 6 cells, then layer 5, 4, 3, and finally 2 [69]. Layer 1 is an exception: it is formed from the MZ cells that were among the first to be created.

Most of these cells will differentiate to form the excitatory cells of the cortex. Many cortical interneu-

rons are formed in the lateral and medial eminence (future basal ganglia), and migrate tangentially into the cortex; nevertheless, many of them follow the same route than the excitatory cells [70].

There are two major migration patterns for the excitatory cells [71–73]. The first one is soma translocation that occurs in the early stages : a cell which contacts both the inner and outer surfaces of the neural tubes looses contact with the VZ, and through traction along the process connecting the pial surface pulls its soma radially upward. The other pattern is the glial guided mode, where young neuroblasts use the radial glial cells process as scaffold to migrate along. They stop their journey when they reach the edge of the MZ.

The stopping mechanisms are not absolutely clear. It is known that the protein reelin, produced mainly by the Cajal-Retzius cells in VZ is necessary for the migrating neuroblast to detach from the radial glial process. The absence of this protein leads to severe disorganization of the cortical architecture [74][2].

The dendritic and axonal arborization is already initiated during radial migration, when many pyramidal cell precursors extend a leading process toward the pial surface and leave behind a long and thin trailing process that often can be traced back to the VZ, and that can extend in the direction opposed to the cell body's movement. Once the cell has reached it's final position, the leading process transforms into a dentritic-like structure (the axis of the leading process giving the direction for the apical dendrite), whereas the trailing process becomes the major shaft of the axon [79]. Once the cell body is at its proper position, it becomes rounder and the differentiation goes on with the elongation of neurites (axon or dendrite).

The tip of a growing neurite contains a motile and sensitive structure called the growth cone (GC) [80]. The GC has long projections called filopodia, classically compared to fingers, between which is a veil-like structure called lamellipodia. The filopodiae actively extend and retract, until one of them is stabilized, defining the direction of growth. It may happen that two filopodiae become permanent, which leads to a splitting and thus a bifurcation of the neurite[3].

The decision to extend, to turn, to retract or to branch is made depending on specific chemical and non-

[2]Several experiments have shown that reelin is not a stopping signal -at least not the only one. (1) If this protein is injected in the ventricle of a deficient embryo it corrects to a certain extend the phenotype, even if the protein is not expressed at the right place [75, 76]. (2) Post mitotic ferrets embryonic cells taken from an E29 animal (when neurons normally destined for layer 6 and subplate are produced) are transplanted in older brains where layer 2/3 neurons were being generated [77], the transplanted cell usually migrates only to L6. Which indicates that their is also a stopping mechanism based on neighboring cells. (But when progenitors are transplanted during the S-phase, the majority of their daughter neurons switch their normal laminar fates and migrated into layer 2/3 . This indicates that environmental factors can modify laminar fate in progenitors just before mitosis but not in migrating neuroblasts.) (3) If a neural precursor taken in an older embryo (E49 ferret), designed to become a L2/3 neuron, is transplanted into a younger animal (E32, when L5 cells are being produced), the precursor maintains its elongated morphology, ), slowly following the top of the cortical plate until the host cells start to form L2/3 [78].

[3]In cortical neurons, bifurcation is probably a less common branching mode than a secondary side branching on an already existing shaft [81, 82])

specific physical cues[4]. The classical families of guidance molecules are Ephrins, Netrins, Semaphorins and Slits [84,85]. More recently, several other molecules have been discovered to play a role in axon guidance, like growth factors, early embryonic morphogenes (like Sonic Hedgehog) or even neurotransmitters [86,87]. An active GC receives simultaneously various different cues, potentiating, modulating or inhibiting each others. The final integration depends on the inner state of the GC, like the Ca concentration or the expressed receptors. The same substance can have opposite effects on the same GC within a few hours [88].

To form a functional network, neurons must then connect by forming synapses. This process can occur during the axonal elongation, or at a later stage, by establishment of connections between protruding structures on axonal and dendritic shaft. It seems that the process can be initiated either by the pre- or the post-synaptic cell [89].

Finally we have to mention that numerous neurons are eliminated by apoptosis, and in surviving cells branches can retract. Often a structure with numerous connexions is stabilized, and thus less likely to be eliminated. This closes the loop: structure growth $\rightarrow$ connection $\rightarrow$ function $\rightarrow$ stabilization.

[4]Historically, the first observation of a guidance mechanism was made by RG Harrisson in 1910 when he observed that cultured neurons tended to follow glass defect of the dish (this property is still used to obtain specific architecture of in vitro networks [83]).

# Simulation Environment

# Chapter 2

# Physical methods

## 2.1 Organization of the extracellular space

### 2.1.1 Neighborhood relation

In our simulator, neurons are composed of discrete physical components such as spheres (somata) and cylinders (neurites), each located at a particular point in 3D space, where they interact locally with one another, simulating the physical and biological processes occurring in the tissue (Figure 2.1). Each evaluation for a possible interaction between object $i$ and $j$ has a computational cost. Clearly, to evaluate each possible pair $(i, j)$ at each time step would become prohibitively expensive as the number and complexity of the neurons grow. Instead, we maintain for each object a list of neighboring objects with which it might interact. This list is updated when an object moves, or when an object is added or deleted from the space.

To define this neighborhood relation we use a 3D *Delaunay triangulation* [90]. Given a set $P$ of points in 2D, a triangulation $T$ is a collection of non-overlapping triangles whose vertices coincide with the members of $P$, that covers the convex hull formed by $P$. The points and the edges of the triangle define a graph structure. Two points are defined as neighbors if and only if there is at least one triangle $t \in T$ of which both are a vertex, i.e. if they share a common edge in the graph. The Delaunay triangulation is a special triangulation, defined by the condition that no point of $P$ is inside the circumsphere of any triangle of $T$. In 3D, the method generalizes to the *Delaunay tetrahedralization*, where a set of points in space defines a set of tetrahedrons (for simplicity, we will nevertheless use the term triangulation even in the 3D case). In our framework, each discrete object is associated with a vertex in a 3D triangulation.

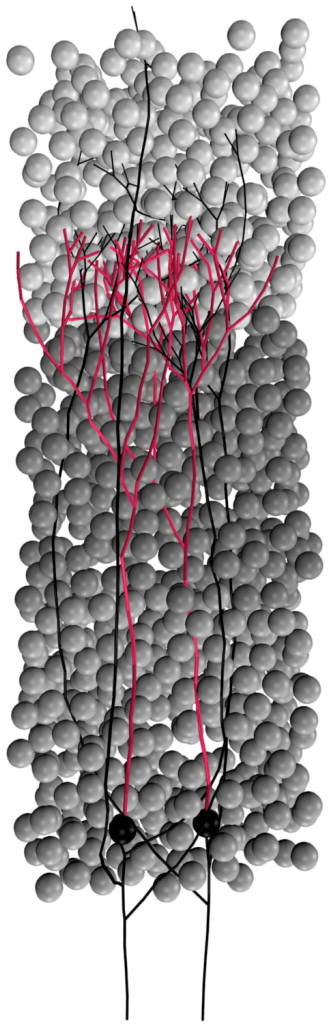

**Figure 2.1. Typical CX3D simulation.** The figure shows the result of a simulation in which two neurons extend dendritic (red) and axonal (black) arbors in a dense cortical column, according to the model specification described in Figure 7. The physics engine prevents a branch from passing through another cell. (Half of the cells in the column were removed for better visualization). The 3D rendering was obtain by exporting the result of the CX3D simulation into the free program Blender (http://www.blender.org). The mesh used for the rendering was created with the free java-based software ImageJ3DViewer (http://www.neurofly.de/ImageJ3DViewer).

CX3D uses the package Dyna3D written by Goehlsdorf (http://www.ini.ethz.ch/∼dennis/).

If the cell density is very low, it might happen that two physical objects far apart are considered as being neighbors, just because there is no other object between them. In this situation, for computational reasons, the user might want to add additional 'empty' vertices to the triangulation, so that physical interactions between pairs of remote objects are not evaluated (Figures 2.2AB).

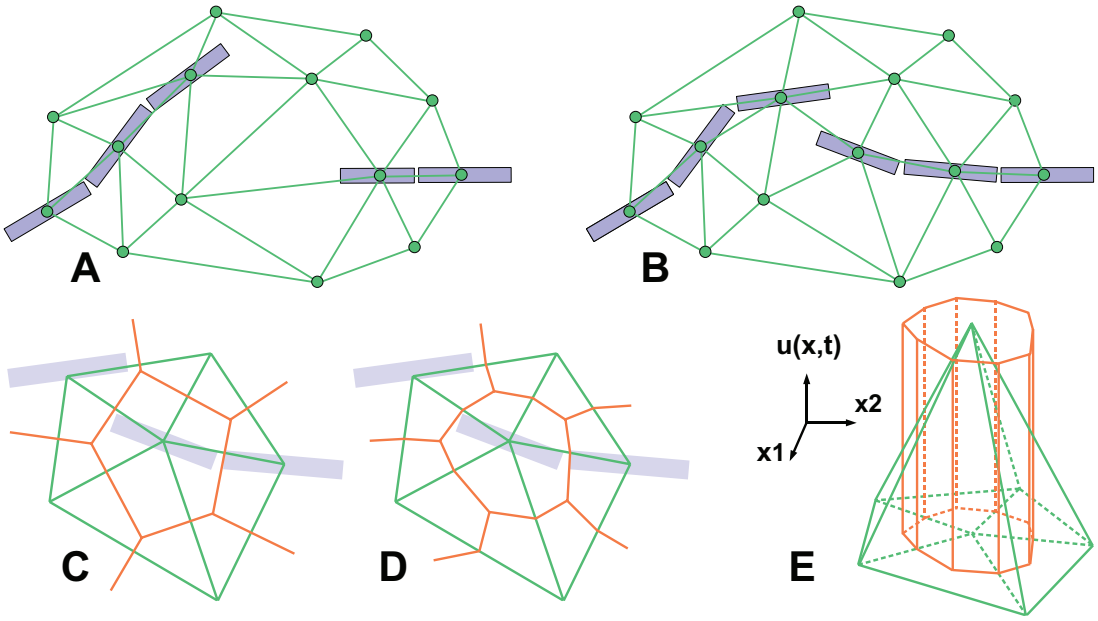


**Figure 2.2. 2D illustration of the Delaunay triangulation and its dual graphs. (A)** Each discrete physical object (blue) in the simulation is linked to a vertex in the Delaunay triangulation (green). Additional vertices represent empty regions of space. Two objects are considered as being neighbors when their vertices are linked by an edge. **(B)** When existing objects move or are deleted, or when new objects are created, their associated Delaunay vertex is automatically moved, deleted or inserted, and the triangulation locally updated. **(C)** The *Voronoi graph* (orange) is an example of a dual graph used to define a vertex-centered volume decomposition based on the Delaunay triangulation. The volume around each vertex contains every point in space that is closer to this vertex than to any other. **(D)** Another dual graph : the *median dual graph* is the set of lines joining the centroids (or barycenters) of all edges and triangles adjacent to a vertex (in 3D: all the edges, triangular faces and tetrahedrons adjacent to a vertex). **(E)** In the finite volumes method, for a given substance, only the average concentration $u_i(t)$ over a domain $V_i$ is known. The total quantity $Q_i(t)$ of the substance inside the domain is equal to $u_i(t) \cdot V_i$ (the volume of the orange column). If the domain is defined by the median dual graph, a linear vertex-centered function with peak of $u_i$ contains exactly the same quantity (volume of the green pyramid). This representation is extremely convenient when we have to interpolate the concentration outside the vertices.

### 2.1.2 Diffusion processes

For the simulation of diffusion, we use an approach similar to the finite volume method [91]. The extracellular space is decomposed into small non overlapping domains. When a physical object secretes a certain quantity of a signaling substance, the concentration of this substance increases in the domain containing this object. Let $i$ and $j$ be two compartments with respective volume $V_i$ and $V_j$, containing the amount $Q_i$ and $Q_j$ of a given substance (hence the concentrations are $u_i = Q_i/V_i$ and $u_j = Q_j/V_j$). If they are in contact, Fick's first law tells us that the net flux $J_{i \to j}$ (in units of quantity per time) going from $i$ to $j$ is:

$$J_{i \to j} = D \frac{S_{ij}}{d_{ij}} (u_j - u_i),$$

where $D$ is the diffusion coefficient of the substance, $S_{ij}$ the area of contact between the compartments and $d_{ij}$ the distance between their centers.

A first approach would be to multiply the flux $J_{i \to j}$ by the simulation time step $\Delta t$ to compute the quantity transfered from $i$ to $j$ during the time step, to subtract it from $Q_i$ and add it to $Q_j$. The new concentrations could be found by dividing the new quantities by the respective volumes. Using this formula in our simulation is equivalent to the Euler explicit method. But it comes with a very high risk of overshoot if the time steps are too large, especially in our case with an irregular decomposition of space. It is thus preferable to solve analytically the diffusion between each pair of neighbors: Remembering that $Q_x$ and $u_x$ vary with time, we obtain the following ordinary differential equation:

$$\frac{d}{dt} Q_i(t) = D \frac{S}{d} (u_j(t) - u_i(t)) = D \frac{S}{d} (\frac{Q_j(t)}{V_j} - \frac{Q_i(t)}{V_i}).$$

To get rid of the dependance on the quantity in the compartment $j$, we define the total amount $T = Q_i + Q_j$ that is time-invariant. We can now solve the equation above and obtain:

$$Q_i(t) = K e^{-mt} + \frac{n}{m}$$

with $m = D \frac{S}{d} (\frac{1}{V_j} + \frac{1}{V_i})$ and $n = D \frac{S}{d} \frac{T}{V_j}$, and the integration constant $K = Q_i(t_0) - \frac{n}{m}$.

### 2.1.3 The median dual graph

The Delaunay triangulation that we use for near-object detection already provides us with a decomposition of space in discrete volumes (the tetrahedrons). But since all substances are produced and probed at the vertices of the triangulation (where the cell elements are located), it makes sense to use a *dual graph*, i.e. another decomposition containing the Delaunay nodes in the center of its volumes. The most popular graph with this property is the Voronoi graph (Figure 2.2C), but for computational reasons we use the median dual graph (Figure 2.2D). Firstly because it is not necessary to compute the boundaries of a domain to compute its volume (it's simply one fourth of the volume of the adjacent tetrahedrons). Secondly, because if we consider that the average concentration $u_i(t)$ for domain $i$ given by the finite volumes method corresponds to the real concentration at the vertex position, and that we use linear interpolation

between the vertices to define the concentration elsewhere, we get a better numerical approximation with the median dual graph (Figure 2.2E).

To define the gradient on the Delaunay vertices, we recall that the directional derivative of the concentration $u$ at the point $\mathbf{x}_i$ along the unitary vector $\hat{\mathbf{e}}$ is equal to the dot product of $\hat{\mathbf{e}}$ with the gradient of $u$ at $\mathbf{x}_i$ :

$$D_{\hat{\mathbf{e}}}u(\mathbf{x}_i) = \hat{\mathbf{e}} \cdot \nabla u(\mathbf{x}_i)$$

We can approximate the directional derivative at $\mathbf{x}_i$ along a vector pointing to any neighbor vertex $\mathbf{x}_j$ by taking the difference of the two concentrations divided by the distance between them. With three different $\mathbf{x}_j$, we obtain a system of three equations that we solve to find the three components of the gradient at $\mathbf{x}_i$ :

$$\nabla u(\mathbf{x}_i) \cdot (\mathbf{x}_j - \mathbf{x}_i) = u(\mathbf{x}_j) - u(\mathbf{x}_i), \text{ for } j = \{1, 2, 3\}.$$

The smaller the volumes of the dual graph are, the better the precision of the diffusion simulation. This is another justification for having additional vertices added to the Delaunay graph even in absence of physical objects at that location

Figure 2.3A shows a test system introduced to illustrate the performance of our simulator on various aspects of diffusion. It consists of 500 vertices randomly distributed into a $200 \times 200 \times 200 \mu m^3$ cubic volume. The points are triangulated, with the median dual graph defining 500 volumes surrounding the vertices. Inside each discrete volume, we place a precise quantity of three diffusible substances in order to get a desired concentration, varying with the position of the vertex along one spatial dimension: The concentration profile of chemical R (red) is a step function, of G (green) a linear function and of B (blue) a cosine. Figures 2.3B and 2.3C show the evolution of the concentration profiles over time due to diffusion.

### 2.1.4 Chemical reactions

In addition to diffusion and secretion by cells, the substances in the extracellular space are subject to concentration changes due to degradation and possibly other chemical reactions. Degradation is processed together with diffusion (a diffusion and a degradation constant can be specified for each extracellular substance). To introduce chemical reactions, the user has the possibility to define changes of concentrations that are applied at each time step on each discrete volume of the extracellular space.

As an illustration, we implemented in our test system the reaction $R + G \rightleftharpoons_{k_{-1}}^{k_1} B$ (with $k_1 = 10$ and $k_{-1} = 0.5$), which corresponds to the combination of one molecule of the red and one molecule of the green substance forming one molecule of the blue substance, by applying the following concentration changes in each volume at each time step:

$$-\frac{d[R]}{dt} = -\frac{d[G]}{dt} = \frac{d[B]}{dt} = k_1[R][G] - k_{-1}[B].$$

Figure 2.3D and Figure 2.3E show the result without and with concurrent diffusion repectively.

### 2.1.5 Influence of grid deformations on the concentration profile

Modifications of the Delaunay mesh have dramatic effects on the dual graph that we use to numerically solve diffusion. Thus we needed to incorporate a mechanism to automatically redistribute the different quantities of substances after each operation on the triangulation (physical object displacement, duplication or removal). The two major requirements are to preserve the concentration profiles, and to ensure mass conservation. Consider the case where the Delaunay vertex at position $\mathbf{x}_i$ moves. If we didn't update the quantity of substance located inside the surrounding volume, moving the point would result in substance transport. Our update mechanism consists of two phases: first we interpolate the concentration $u'_i$ of the chemical at the new location $\mathbf{x'}_i$ of the moving vertex, and modify the quantity in the newly formed volume $V'_i$ to obtain this desired concentration, i.e. define new $Q'_i$ so that $Q'_i/V'_i = u'_i$. Then we compensate for total mass conservation by multiplying the concentrations and the quantities in the surrounding volumes by the ratio between the total quantities before and after the displacement. Similar update mechanisms are used for vertex insertion or removal.

The procedure is tested in our bench test by moving three inner vertices 100 times (Figure 2.3F: The displacement is a random 3D vector of less than $5\mu m$, with a re-centering mechanism to ensure that the points stay inside the convex hull of all other points). This minimally disruptive procedure allows for gradient ascent even in the extreme case where all physical objects are moving (Figure 2.3G).

## 2.2 Mechanical properties of neurons

The complex shape of neurons, composed of dendritic and axonal arbors, makes the computation of their mechanical properties and interactions a difficult task. Following a technique that is used commonly in mechanical engineering and virtual reality contexts [92, 93], neurons in CX3D are composed of chains of

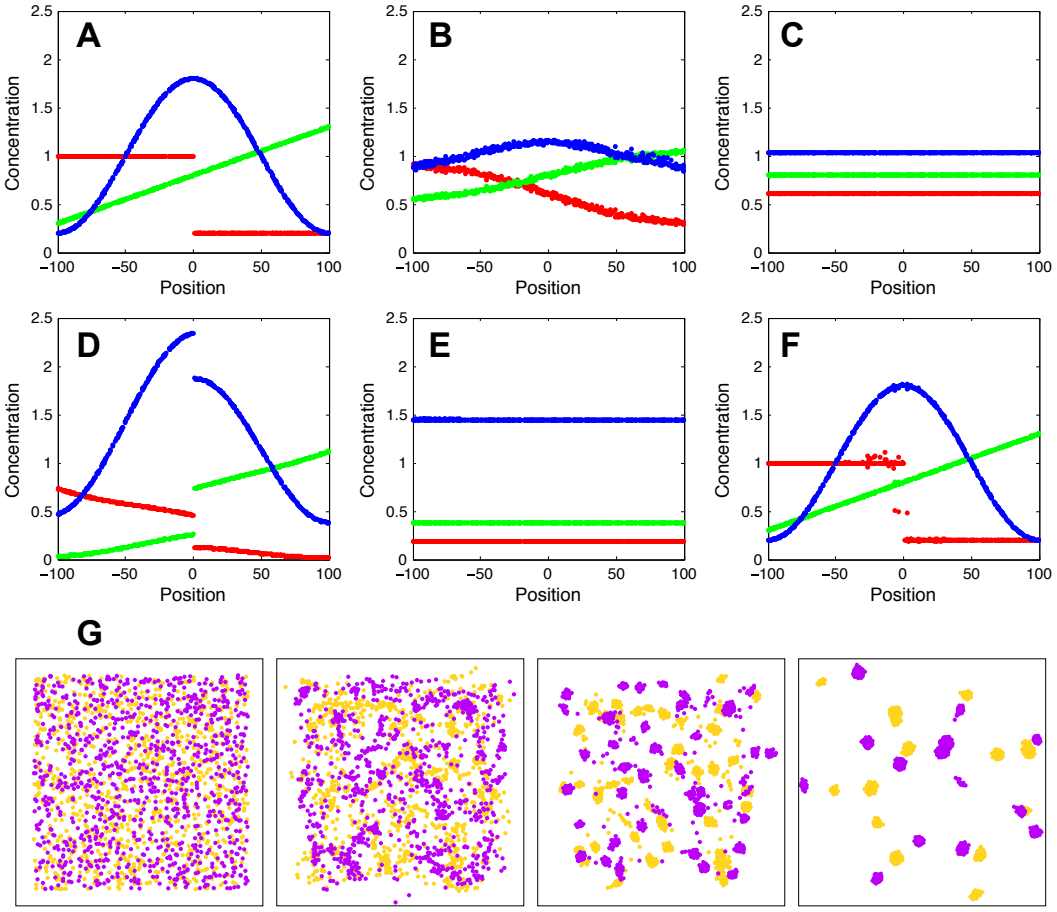


**Figure 2.3. Test system for diffusion and chemical reactions (A)** Initial chemical concentrations in the discrete volumes created by the triangulation of 500 vertices randomly placed inside a cube. The concentration of three substances define a particular profile, dependent on the volume's central vertex location along the horizontal axis (red substance: step concentration profile; green substance: linear profile; blue substance: cosine profile). **(B)** Evolution of the concentration profiles due to diffusion after 50 simulation time steps. **(C)** Concentration profiles after 500 simulation time steps. **(D)** Concentration profiles after 500 simulation time steps due to the chemical reaction $R + G \rightleftharpoons B$ (see text), without diffusion. **(E)** Concentration profile after 500 steps of the same chemical reaction with diffusion included. Compare with diffusion without reaction in (C). **(F)** Minor alteration in the concentration profiles after 100 sequences of movements of three central vertices, without diffusion or reaction. **(G)** Use of diffusion in simulation: 1000 yellow cells and 1000 violet cells, secreting respectively 'yellow' and 'violet' diffusible cues, are distributed randomly in a 3D volume. They aggregate by following the gradient of their cell-type specific cue (results after 0, 300, 800 and 6000 time steps).

springs and masses in series to provide structural integrity and propagate tension. Spherical and cylindrical wrappers enclose the spring-mass chain to confer volume on the cell and define spatial interactions between neighboring objects. Each wrapper is an independent object containing a single point mass. At each time step, it computes the local forces on its point mass and moves it accordingly.

Somata are defined by a sphere with a central point mass, whereas neurites are composed of cylinders, each containing one spring and one point mass at its distal end (Figure 2.4). The disadvantage of this configuration is the asymmetry of the cylinders. The substantial advantage, on the other hand, is that we can neglect rotations of cylinders. Indeed each one is responsible for moving only its distal end (where the point mass is located), whereas its proximal end is defined by the position of its attachment point on the proximal discrete object. During neurite extension, some cylindrical elements are elongated. If their length exceeds a specified threshold, they split into two elements. Similarly, in case of retraction, short cylinders fuse. By this mechanism we ensure the suitable discretization of the cell. This discretization also permits intracellular diffusion. The simulation is performed in a similar way as for extracellular diffusion, but the volumes are defined by the cylinders and the sphere composing the neuron, and substances flow along the chains of elements regardless of the triangulation or its dual graph.

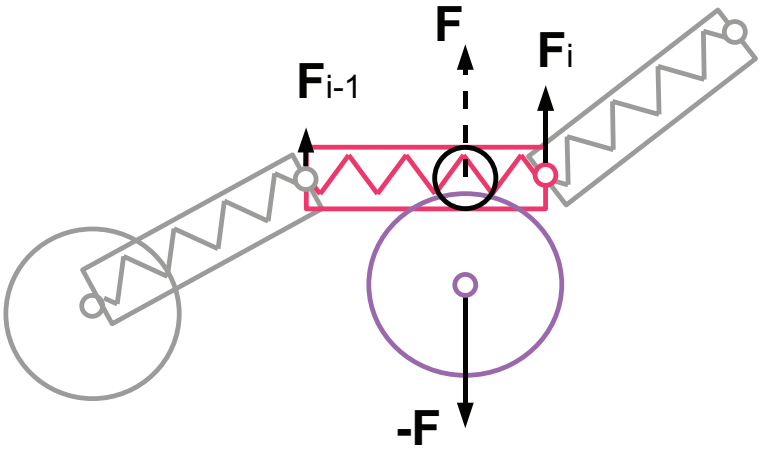


**Figure 2.4. Cartoon representation of the physical discretization and inter-object mechanical forces in CX3D.** Neurons are discretized into small physical objects, composed of a single point mass and a spherical (for the cell body) or cylindrical (for the neurite elements) envelope. The envelopes are used to define inter-object forces when two objects come into close contact. In this example, a cylinder in a neurite (red) and the sphere of an other cell's soma (violet) overlap, which triggers opposite repulsive forces ($\mathbf{F}$ and $-\mathbf{F}$). To determine the repulsion intensity, we define a virtual sphere (black circle) on the cylinder. The forces are proportional to the overlap of the virtual sphere and the soma sphere. Spheres have a central point mass, and the force is directly applied on it. Cylinders have their point mass at the distal extremity, so only a fraction of their inter-object force is applied on it ($\mathbf{F}_i$), while the rest is transmitted to the proximal element's point mass ($\mathbf{F}_{i-1}$). In addition, cylindrical elements contain a spring which is always attached to another proximal element, and propagates tension along the chain of point masses.

### 2.2.1 Forces

Three different types of forces can be applied to each point mass. The first type arises from the interaction between the physical objects when they come into close contact. The second type is the internal tension arising when a neurite is stretched, which is modeled by the springs connecting the masses. This internal tension both influences, and is influenced by, metabolic growth. The third type of forces represents the active movement of cell elements. It follows from the biological properties of the model specified by the user.

#### Inter-object forces

Cells in a tissue have strong resistance to compression. They also have adhesive properties. Consequently they are conveniently modeled as a granular medium with additional bindings [45, 94]. The physical interaction between two spherical somata is then a function of their diameter, their relative distance, and possibly their expression of adherence molecules.

It is possible for users of CX3D to define their own cell-cell interaction function. By default a modified version of [95] is used, in which the force from sphere $s_i$ onto sphere $s_j$ contains a repulsive component (preventing two cells to overlap) and an attractive component (representing the integrity of the tissue and forces resulting from cell adhesion molecules):

$$\mathbf{F}_{ij} = \left( k\delta - \gamma \sqrt{\frac{r_i r_j}{r_i + r_j}\delta} \right) \hat{\mathbf{e}},$$

where $k$ is the repulsion coefficient, $\gamma$ the attraction coefficient, $r_i$, $r_j$ the radii of the spheres, $\delta$ the overlap: $\max(0, r_i + r_j - \text{dist}_{ij})$, and $\hat{\mathbf{e}}$ the unitary vector pointing from the center of sphere $i$ in direction of sphere $j$.

The radius $r_i$ used in the previous equation needs not to be exactly the radius of the sphere $s_i$. For instance, to reproduce the different neuron densities observed one can use larger effective radii, which increases the range of interaction and hence pushes cells further apart from each other. Or one can use smaller radii for migrating cells, and thereby model the possible deformations of moving cells that are less perturbed by the surrounding tissue.

In CX3D, we must also consider cylindrical objects, which means that there are in fact three different sorts of interactions: sphere-sphere, sphere-cylinder and cylinder-cylinder. For instance, to compute the interaction between a sphere $s_1$ and a cylinder $c$, we define on $c$ a virtual sphere $s_2$, and then compute

the interaction between the two spheres $s_1$ and $s_2$ as described above.

Cylinders have their unique point mass located at their distal end. So, if the inter-object forces applied to a particular cylinder only affect its own point mass, it means that only one of its extremities will ever move. Therefore, part of this force has to be transmitted to its proximal segment (i.e. the object responsible for the mass situated at the proximal end of the cylinder). This repartition of forces along chains of cylinders is essential for stability (Figure 2.5).

In addition to the attractive component of the inter-object interaction, additional specific adhesive bonds, permanent or transient, can be added between neighboring objects. These links consist of additional springs between two discrete physical objects. Such links can be used, for example, for neurite fasciculation (Figure 2.6), or to stabilize the pre- and post-synaptic cell elements with respect to one another at the location of a synapse.

**Intracellular tension**

Dennerll et al. [15] have reported that neurites show passive viscoelastic properties when stretched with a force smaller than 1 nanonewton. During the 10 first minutes they observed two passive phases: a rapid increase in length and tension, followed by a damped phase. Mechanical models with a spring and a Voigt element (spring and dashpot in parallel) or a Burger element fit these data well (Figure 2.7A). If a larger force is applied for a longer time, a third phase is observed in which the neurite continues to extend while the tension diminishes, sometimes to less than the tension before the application of the perturbing force. This third phase corresponds to active growth, including reorganization of the cytoskeleton and incorporation of membrane components. This phenomenon explains 'towed growth' (growth not generated by the growth cone).

Our model does not differentiate between the two passive phases, since we consider only springs in series, which is a reasonable approximation to neurite passive mechanical properties [96]. The absence of a dashpot is compensated for by the use of the overdamped approximation in the equation for movement (see below). The tension vector due to the internal spring in a cylinder is thus:

$$\mathbf{T}_{internal} = k\frac{a_L - r_L}{r_L}\hat{\mathbf{e}},$$

where $k$ is the linear spring constant of the neurite, $a_L$ the actual length of the spring (length of the cylinder), $r_L$ the resting length of the spring and $\hat{\mathbf{e}}$ the unitary vector aligned with the central line of the cylinder.

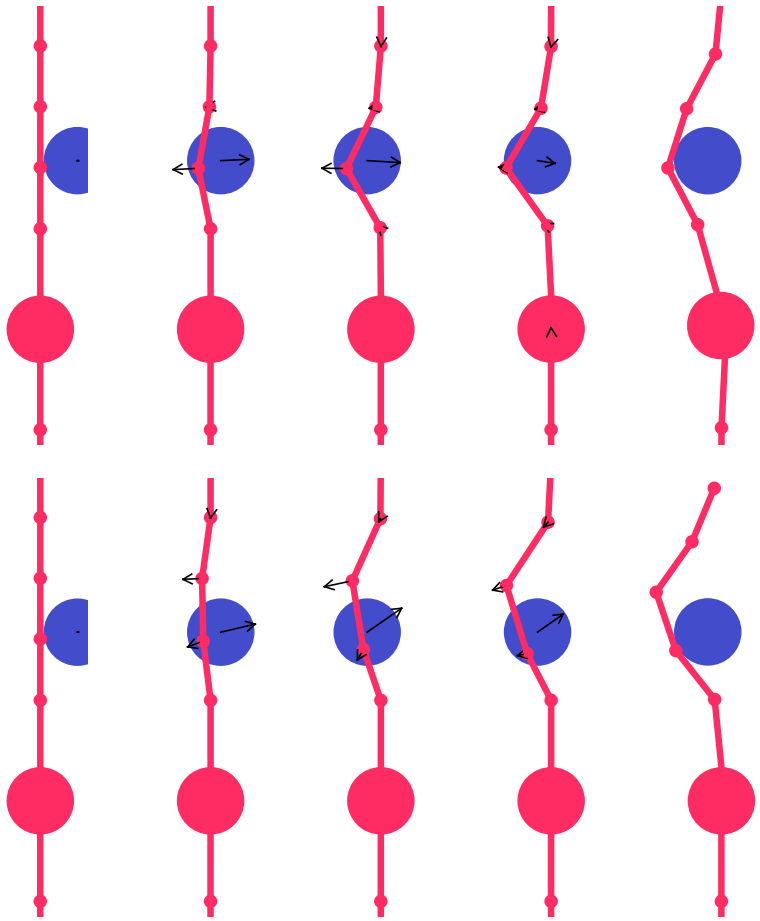

**Figure 2.5. Importance of the transmission of forces to the proximal element in chains of physical cylinders**. A migrating cell (blue) pushes the neurite of another cell (red). This figure shows the outcome of the mechanical forces on the neurite with and without force transmission by cylinders to the previous point mass. After collision, two cylinders of the neurite are being touched by the moving soma. For the upper cylinder the contact is far from its point mass (mainly at its proximal end). The total mechanical force acting on each point mass is indicted with a black arrow (Note that the migrational leftward force of the blue sphere overcomes the rightward reaction force shown in black). (A) If the upper cylinder can transmit some of the force from the blue sphere to its proximal cylinder, most of this force will be applied at the point of contact. (B) In the absence of force transmission, the two physical cylinders will only apply their resulting force on their own respective point masses. The point mass of the upper cylinder, although outside of the contact area, is then displaced, which results in a larger deformation of the neurite.

The metabolic phase is modeled by changing the resting length of the springs; for instance, $r_L \rightarrow r_L + \Delta L$ for elongation, which automatically decreases the tension (Figure 2.7B). As described above, the number of discrete cylinders scales with the length of the neurites. The discretization mechanism is based on the resting length: if it exceeds a certain threshold, the cylinder is split into two, each half retaining the same tension.

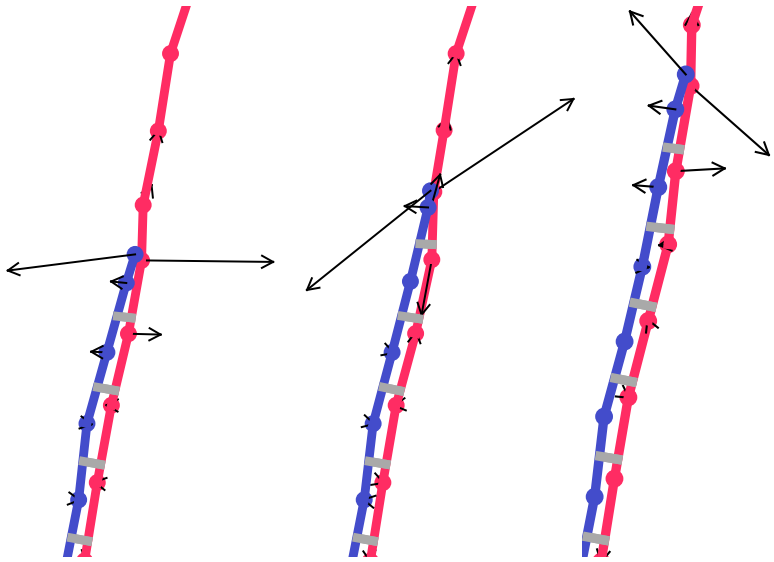

**Figure 2.6. Neurite Fasciculation with additional physical links**. One possibility for modeling adhesion between cell elements is to use additional springs between physical objects. In this simulation the blue neurite uses the pre-existing pioneer red neurite as guide for its growth. Its distal cylindrical segment elongates by following the red neurite. When its length exceeds the maximal length, a new cylinder is inserted which attaches itself to the guiding neurite with an elastic links (grey). The total mechanical force acting on each point mass is indicted with a black arrows.

### Active displacement

The biological properties specified by users often require the active movement of spheres and of neurites' terminal cylinders, for instance in the case of cell migration or neurite extension. In this case, an additional force is applied to the physical objects' point mass. The moving objects do not modify their trajectories ahead of time in case of an upcoming collision. But if after the displacement two objects have come too close (or interpenetrate), a repulsive force is triggered between them (Figure 2.8). This system is as stable as trajectory interpolation, but is less computationally demanding.

In the case of neurite extension, the displacement of the distal point mass must result in an increase in the resting length of the corresponding spring (as in towed growth). It has been shown [97] that the extension rate of an axon is proportional to the tension its growth cone is applying, in the following sequence: movement → stretching (increase in actual length) → increase in tension → active growth (increase in resting length). Although we could reproduce this sequence in CX3D, for computational reasons we usually take a short cut: First the growth cone is moved and then the resting length is set in order to obtain the desired tension. Similar mechanisms are possible for retraction: A reduction of the resting length will induce an increase in tension and then a backward movement of the distal point mass.

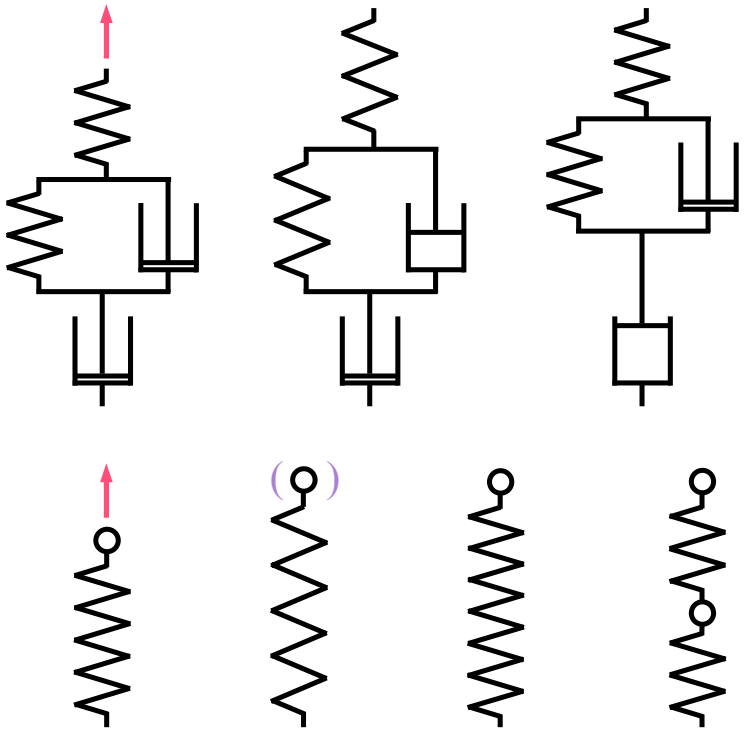

**Figure 2.7. Mechanical models of neurites subject to traction**. (A) According to Dennerll et al. [15], an undamped spring in series with a Voigt element (spring and dashpot) accounts for the purely passive response to stretching. The metabolic growth that follows is represented by an additional dashpot that releases the tension. (B) In CX3D, the neurites are modeled only with springs and point masses; the damping arises naturally from the overdamped approximation used to compute the displacement of point masses (depicted by the purple brackets). Metabolic growth can be easily simulated by changing the resting length of the springs. If the new resting length then exceeds a given threshold, the cylinder is split into two, to ensure optimal discretization.

Alternatively, the point mass can be moved first, and then the resting length is updated to maintain the desired tension.

### 2.2.2 Movement

During the simulation, each discrete physical object evaluates all instances of the three forces applied to it, and sums them to obtain the total force acting on it. If the total force exceeds a certain threshold, the object moves its point mass appropriately (Figure 2.9). For instance, the cylinder $i$ checks if any neighbor in the triangulation is exerting a force $\mathbf{F}_{ij}$ on it, including possible adhesive bonds $b$. It also takes into account the tension in its internal spring ($\mathbf{T}_i$) and in the springs of the daughter cylinders directly attached distally to it, if any (note that a terminal cylinder has no daughters, a cylinder in a

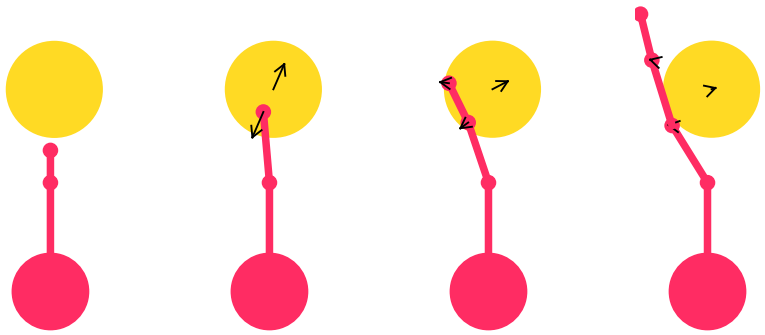

**Figure 2.8. Neurite outgrowth in the presence of an obstacle**. The extension of a neurite is realized by the displacement of its distal point mass. There is no trajectory planning ahead of time to avoid the collision with another cell. Instead, the neurite interpenetrates the other object, inducing a force on both elements. In the example shown in this figure, the yellow cell has a higher adherence (it is fixed to the matrix), and hence the force has no effect on it. On the contrary, the neurite deviates laterally. The total mechanical force acting on each point mass is indicted with a black arrows.

chain has one daughter and a cylinder proximal to a branching point has two daughters). Finally, it might also have some biological movement $\mathbf{M}$ to take into account:

$$\mathbf{F}_{i\,tot} = \sum_{j}^{|Neighbors|} \mathbf{F}_{ij} + \sum_{k}^{|Bonds|} \mathbf{F}_{b_k} + \mathbf{T}_i + \sum_{d}^{|Daughters|} \mathbf{T}_d + \mathbf{M}.$$

In classical mechanics, the equation for movement in a medium with friction is

$$m\ddot{\mathbf{x}} + \beta\dot{\mathbf{x}} = \sum \mathbf{F},$$

where $\ddot{\mathbf{x}}$ is the acceleration, $\dot{\mathbf{x}}$ the speed, $m$ the mass and $\beta$ the kinetic friction. Neuron elements in a tissue have a low Reynolds number (typically $10^{-7}$ for a growth cone [98]), which means that the ratio of the inertial forces to the viscous forces is low. It is then perfectly reasonable to make the overdamped approximation: That is, to assume $m\ddot{\mathbf{x}} = \mathbf{0}$. The consequences of this assumption are (1) that an element doesn't move if it is not currently subject to a force and (2) that the major obstacle to movement is no longer the mass but the friction coefficient. Consequently, physical objects in CX3D move according to:

$$\beta\dot{\mathbf{x}} = \sum \mathbf{F}.$$

In addition, we have a term for static friction that provides a threshold for the initiation of movement. If the total force exceeds the static friction, then the actual movement is computed using the explicit Euler

method: Multiplying the speed with the time step to obtain the actual displacement:

$$\mathbf{\Delta x} = (\frac{1}{\beta}\sum \mathbf{F})\Delta t.$$

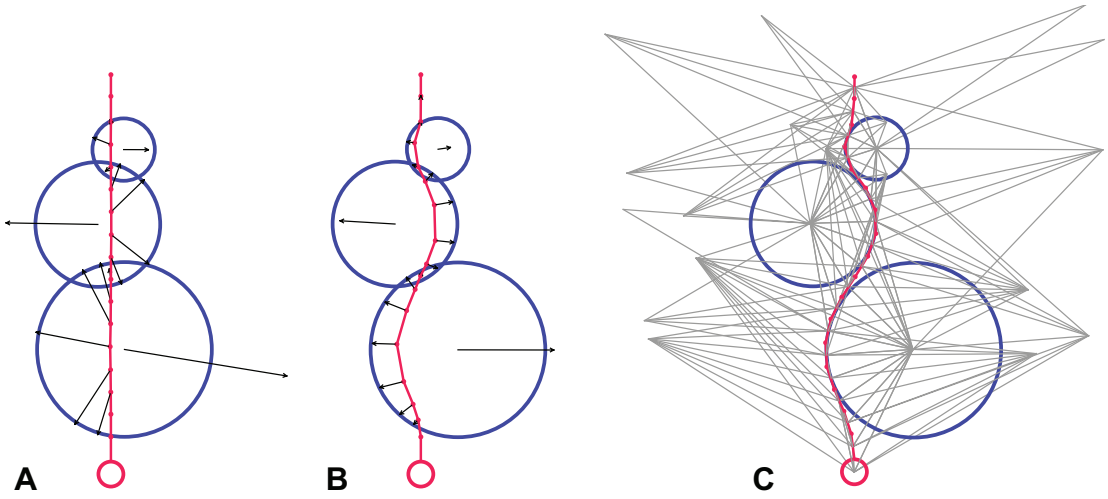


**Figure 2.9. CX3D simulation snapshots demonstrating the mechanical interactions in a test system**, composed of a chain of cylinders attached to a small sphere (red), and three bigger spheres (blue). The total force applied to each point mass is represented by a black arrow. **(A)** We start with the artificial situation where the chain of cylinders and the three spheres are strongly overlapping. This condition triggers a radial force on each of the cylinders' point masses, pushing them outside the spheres. The spheres are subject to the sum of the opposite forces, plus the sphere-sphere interactions. **(B)** After the system has started to relax due to objects being displaced, the magnitudes of the forces start to diminish. The radial movement of the cylinder point masses has resulted in an elongation of the internal springs joining them, which triggers an intra-object force that adds to the component due to the object overlap, and hence results in a less radial total force on the cylinders. **(C)** After complete relaxation, there is no object overlap anymore, and the distances between the point masses of the cylinders chain correspond to the springs resting lengths. Therefore there is no force present anymore. For each discrete object in the system, the link to its neighbors, defined by the 3D Delaunay triangulation is shown (grey). Note the additional vertices that are not corresponding to a physical object.

# Chapter 3

# Implementation and first results

## 3.1 Program architecture

CX3D is written in JAVA because it is an object-oriented language; it benefits from many libraries (for visualization for instance); it does not have to be recompiled to run on different platforms; and it provides methods for distributing processes across multiple computers, which will be crucial for future development.

To design a particular cellular model, users must write modules (small java classes implementing a special interface) that are inserted into the cells to engender their specific functionality. There are two different types of modules: Local biology modules; and cell modules.

Local biology modules represent all the local biological processes. Each one is attached to a particular physical object (one of the spheres or cylinders used to represent the neuron) and reads from it physical data such as current volume, tension, or concentration of an extracellular substance. Likewise, the module sends instructions to the physical object, for example to move, change its shape, or to extend a new branch. For instance, the simplest module for performing chemotaxis would repeatedly execute the following three steps: 1) query from the associated physical object the gradient of an extracellular substance's concentration 2) compute a desired movement 3) transmit a movement direction and speed to the physical object. CX3D would perform the displacement, update the physical values (e.g. define a new length in the case of a cylinder), and update the triangulation. If the movement had brought the object too close to another physical object, a symmetrical force will be applied on both objects at the next time step, possibly pushing them away from one another again. Local biology modules can

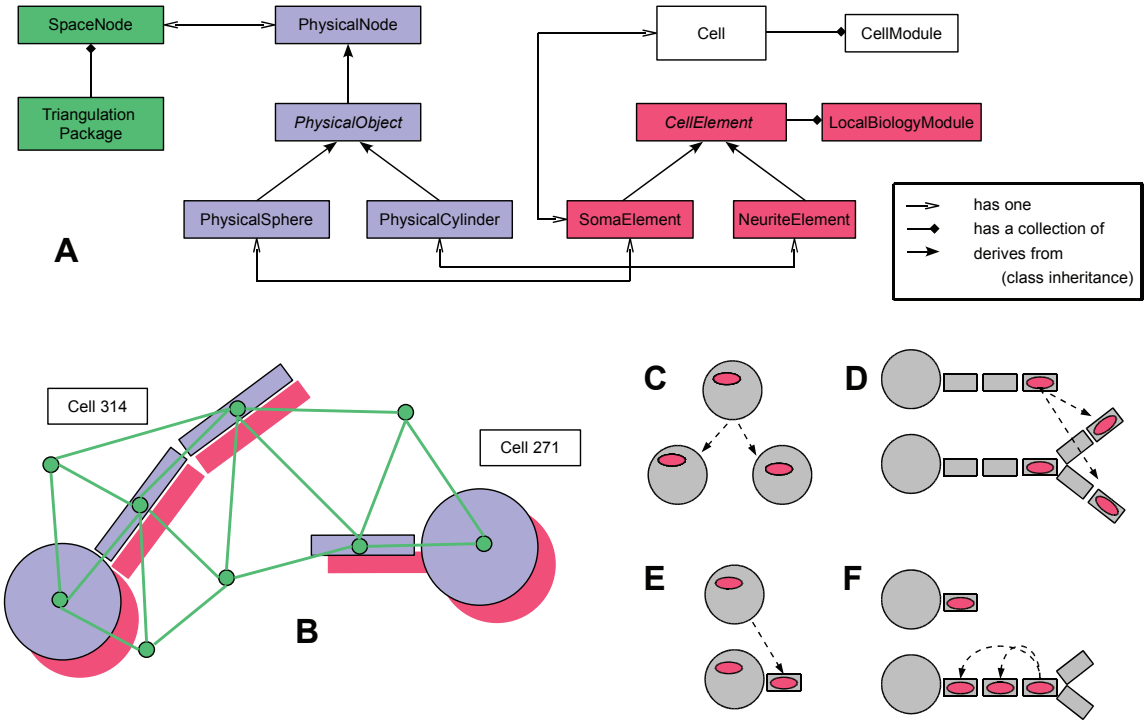


**Figure 3.1. Program architecture of CX3D (A)** Java class diagram and **(B)** cartoon representation of the four abstract layers used in the representation of a cell in the CX3D framework: the Delaunay triangulation defining a neighborhood relation between space regions and physical objects (green); the physical layer containing the classes representing the discrete space regions and the physical objects contained in them (blue); the local biology layer, with biological elements associated with the physical objects (red), and finally the higher level biological properties expressed at the cell level (white). See Appendix for a detailed description of the java classes. **(C-F)** Local biology modules specifying the simulation properties are associated with specific cell elements. When new objects are created, the local biology modules can be automatically copied according to four different schemes: when cells divide, when neurite elements branch, when new neurites are being formed from the soma, and during neurite elongation.

be copied automatically into new discrete cell elements in case of soma division, new neurite extension, neurite branching, or neurite elongation (Figure 3.1C-F).

Cell modules are used to model biological processes affecting the whole cell, such as cell cycles, or gene expression. Because they characterize the entire cell, these modules cannot be linked to any particular spatially located spheres or cylinders that represent the spatial discretization of the cell.

Our software design is modular, keeping a clear separation between the biological processes on the one hand, and the infrastructure needed to run the physics and computationally organize the simulation on the other hand. Four abstract layers are used used in the representation of cells (Figure 3.1AB and Appendix). In addition, CX3D contains several utility packages that are not discussed in this paper.

There are four abstract layers in the representation of a cell in CX3D (Figure 5AB). One purely technical with which the user never interacts, one representing the physics of the simulation, on which the user has to call some methods, and two layers with which the user interacts by writing small modules describing the model's specifications. Each layer correspond to a distinct java package:

1. **ini.cx3d.spatialOrganization**: This layer defines the Delaunay triangulation and median dual graph needed to spatially organize the elements of the simulation, and decompose the extracellular volume. We use the package `DynaD3` developed by Dennis Goehlsdorf (http://www.ini.ethz.ch/∼dennis). Vertices are defined by the class `SpaceNode`, of which each discrete object or space volume has one instance.

2. **ini.cx3d.physics**: The second layer represents the physical processes, both of the extracellular matrix (extracellular diffusion) and of the neurons (mechanics and intracellular diffusion), for which we use instances of the `PhysicalSpace` class, and sub-classes of the abstract `PhysicalObject`, respectively. To have the latter derived from the class defining the extra-cellular matrix volumes ensures that any object in the simulation, as soon as it is instantiated, automatically comes with a minimal definition of the space it occupies. To embody the neurons in the simulation framework we discretize them into small spheres (for the somata) and cylinders (for the neurite segments) with the classes `PhysicalSphere` and `PhysicalCylinder`. They contain the methods needed for the simulation of the mechanics and offer an interface for communication with the biological modules so that the physical shape of the neurons can be modified by growth, branching, retraction etc.

3. **ini.cx3d.localBiology**: The third layer specifies the local biological properties of the simulation, namely the behavior of the spheres and cylinders, with the classes `SomaElement` and `NeuriteElement` respectively (both subclasses of the abstract `LocalBiologyObject`). Instances of these are always associated with a particular `PhysicalObject`. These instances contain modules written by the users to define the specific rules governing the behavior of each discrete object in the model he wants to simulate. These modules are classes that implement the `LocalBiologyModule` interface.

4. **ini.cx3d.cell**: The fourth and last layer defines the higher level biological processes, influencing the whole neuron. As for the local biology level, it is composed of modules that the user can write, implementing a special interface (`CellModule`). These modules are contained in the class `Cell`, of which there is only one instance per neuron.

Finally, the user should be familiar with the package **ini.cx3d.Simulation**, which contains two important classes:

`ECM` contains a list of all the objects currently active in the simulation (instances of the classes described above). This class is also used for adding supplementary vertices to the triangulation, to define chemicals or chemical reactions, and to add boundary conditions.

`Scheduler` contains methods to execute the simulation. That means that it calls the `run()` method of each object. Consequently, the physical objects process diffusion, compute their mechanical interactions and move accordingly. The local biology objects and the cells run all their modules (and thus the models are executed). The triangulation, on the other hand, is not run by the scheduler but only updated in case of vertex displacement, removal or insertion. The first time that the scheduling methods are executed, a GUI window appears, and graphically displays the physical objects.

## 3.2 Sequence diagrams

In this section, we present the sequences of calls and events during the execution of some crucial methods. The following examples are chosen because they are the most common in CX3D. This section can easily be skipped by a reader not interested in technical details of implementations.

### 3.2.1 Cell division

As we have seen, an instance of the class `Cell` is linked to an instance of `SomaElement`, that in turn is linked to a `PhysicalSphere` that (is a subclass of `PhysicalNode`, and as such) contains a `SpatialOrganizationNode`. During a cell division, the four layers have to divide (Fig. 3.2).

When a `Cell` divides, it first creates another instance of a `Cell`, by calling it's `clone()` method. Inside the `Cell constructor`, the new `Cell` is added to `ECM`'s cells list. Then the `Cell` duplicates it's `SomaElement` with a call to `SomaElement.divide()`. As earlier, inside the constructor for this latter class, the new object is registered to ECM. Now the process propagates to the physics level. The mechanism is slightly different in the sense that the Constructor for PhysicalSphere doesn't register itself to ECM; this is done later in `PhysicalSphere.divide()`[1]. The same method requests a new node from the triangulation. Finally, the new instances are linked (`PhyscalSphere` with `SomaElement`, `SomaElement` with `Cell` and reciprocally). What the diagram doesn't show, is that the `CellModules`, `LocalBiologyModules`, `Substances` are also taken care of.

[1]The reasons is that it is dangerous to have a `PhysicalObject` inside `ECM`'s lists before it is integrated to the triangulation.

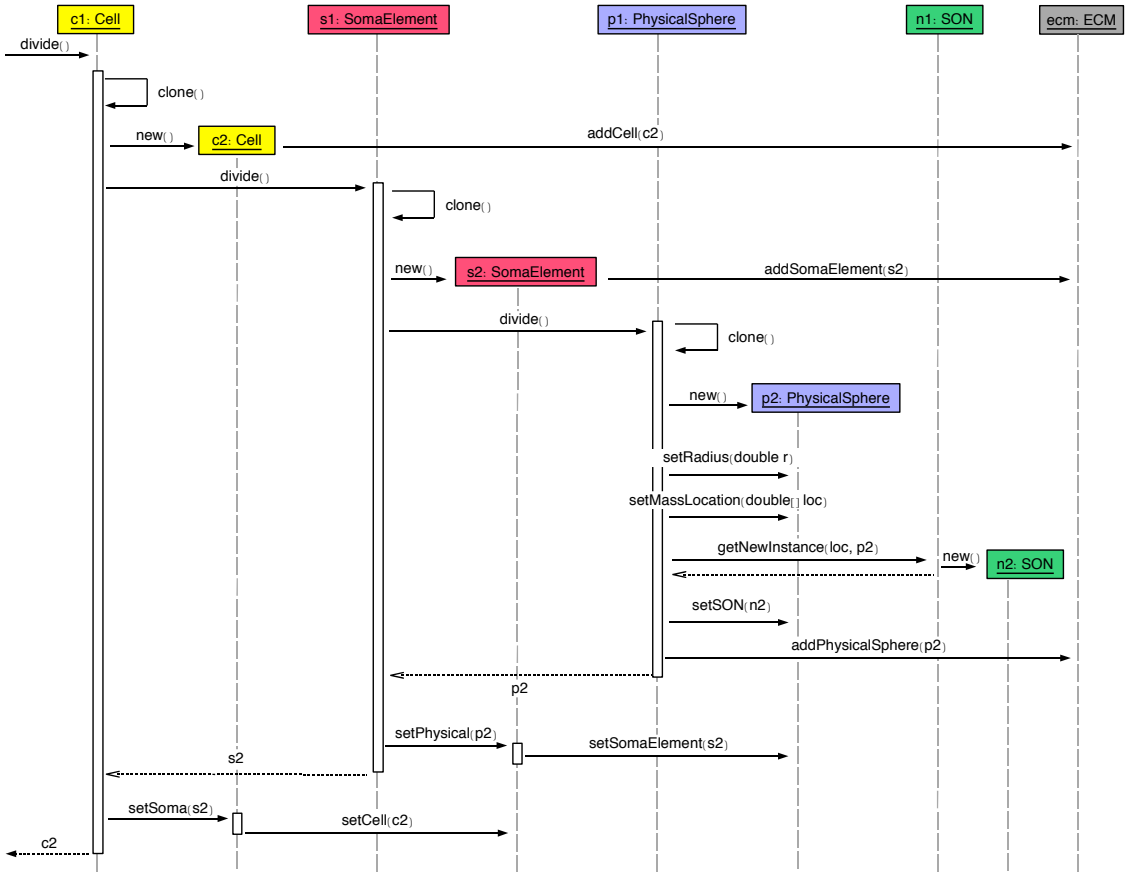


**Figure 3.2. Sequence diagram 1 - Cell.divide()**. During a cell division, objects in the four abstract layers have to be copied, and all references are set correctly.

### 3.2.2 A local biology module defining a movement

In Fig. 3.3, an instance of a class implementing the `LocalBiologyModule` interface guides the movement of a cell compartment up a gradient of extracellular substances. It first asks the `PhysicalObject` (associated with its `CellElement`) for the concentration gradient. Based on this information, it computes a direction and a speed, and moves the object. To compute the concentration gradient for a given `Substance`, the `PhysicalNode` needs to know the position and the chemical concentration contained in three other `PhysicalNodes`. Therefore, it calls it's `SpaceNode` to get the list of the `PhysicalNode` it is connected to. Then for three of them it makes a call to get the concentration of the chemical, and the position. (Note that the position is stored inside the `SpaceNode`, and not inside the `PhysicalNode`). Based on these informations, a gradient is computed by solving a simple linear system.

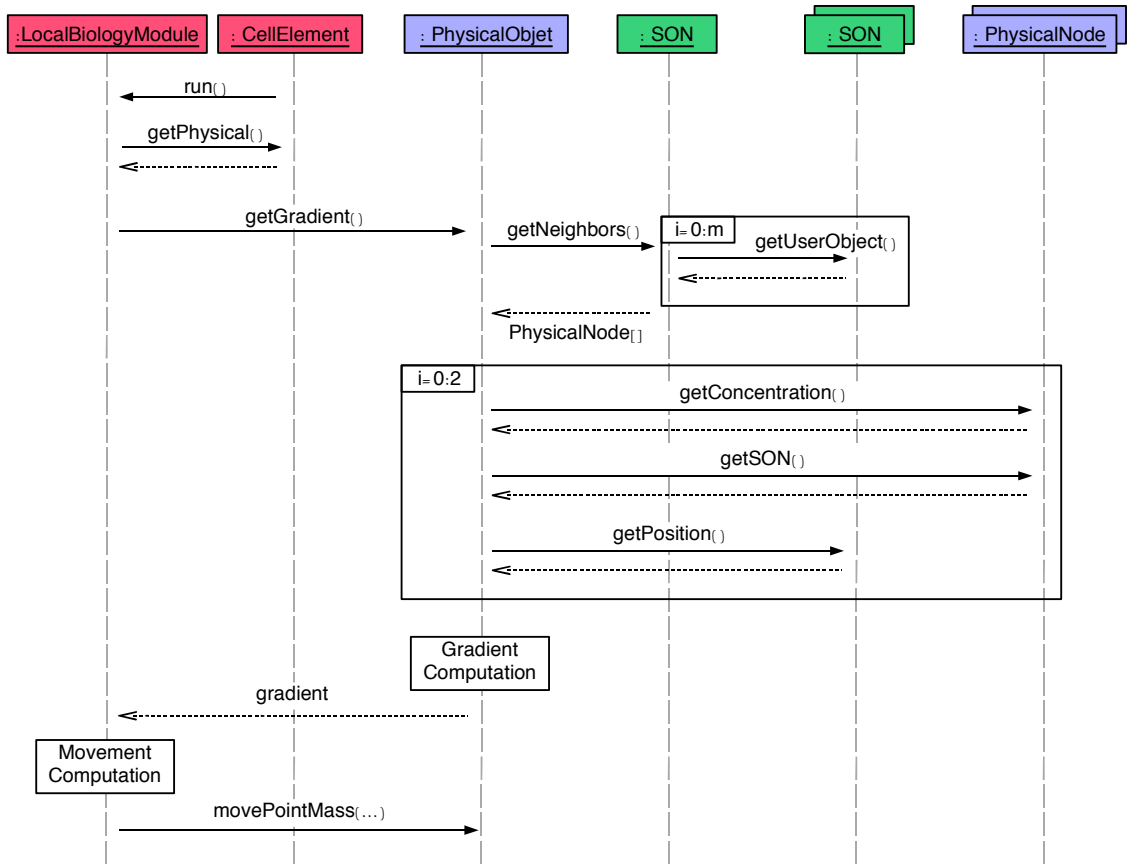


**Figure 3.3.** Sequence diagram 2 - Local biology module). A `CellElement` runs a `LocalBiologyModule` it contains. The module gets a reference to the `PhysicalObject` associated with the `CellElement`. It then asks for the gradient, which is computed by the `PhysicalObject` (based on the concentration value in three neighbors, the references of which are communicated by the Delaunay vertices). Based on the gradient, the module computes a direction and a speed, and then moves the physical object.

### 3.2.3 The passive movement due to physical interaction

Each `PhysicalObject` has a single point mass. All the forces applied to the `PhysicalObject` are applied to this unique point mass, and if it exceeds the friction threshold, the point mass moves accordingly. Here we look at the `runPhysics()` method of a `PhysicalCylinder` (Fig. 3.4).

First the intra-cellular interactions: it computes the effect of its internal spring on the point mass. To this force, we add the force applied by the daughter cylinders (if they exist, you call their `getForceFromDaughters()` method). Now we look at inter-cellular interactions: the `PhysicalCylinder` requests from its `SpatialOrganizerNode` a list of all the neighboring `PhysicalNodes`. If they contain a `PhysicalObject`, the force between the `PhysicialCylinder` and the neighbor depends on their overlap. This is translated into a force by

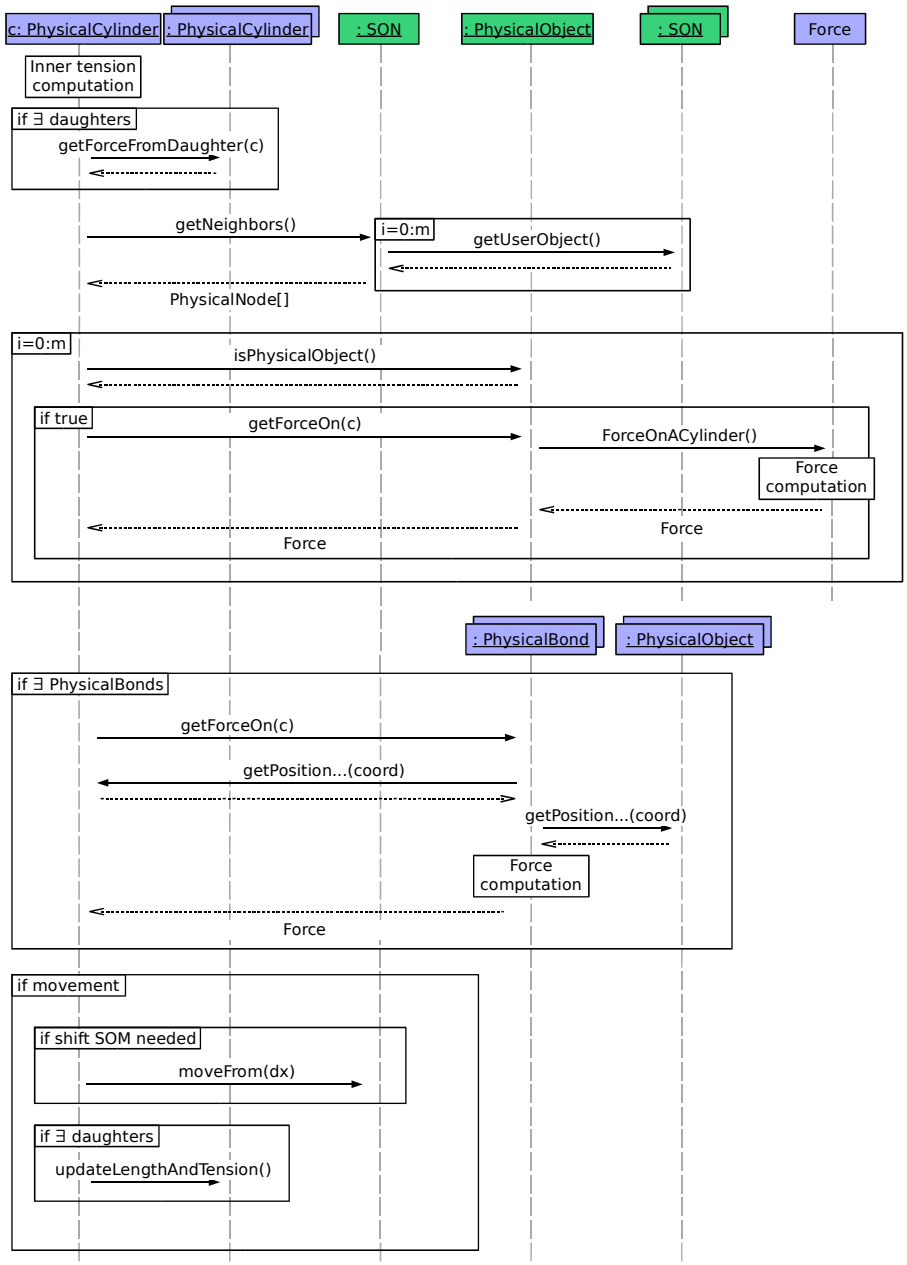


**Figure 3.4. Sequence diagram 3 - PhysicalCylinder.runPhysics()**. This method computes all the forces applied to (the point mass of) a `PhysicalCylinder` : its internal spring, the spring of the potential daughters, the neighbor `PhysicalObjects` that might push it, and the adhesion forces maintaining it close to other objects. Finally, if the norm of the force is important enough, a movement is performed. In which case, the `PhysicalObject` has to inform its `SpaceNode` and its daughters

the static methods of the abstract `Force` class. The result is added to the force applied to the point mass. The last component mechanism generating a force are the `PhysicalBond`, i.e. a spring fixing our `PhysicialCylinder` to another `PhysicalObject`. For all `PhysicalBond` present in the list inside the

`PhysicalCylinder`, we ask to computing a force. This depends on the relative distance of the two points where the `PhysicalBond` is attached on its `PhysicalObjects`. In summary, we have :

$$\vec{F}_{tot} = \vec{F}_{internal} + \sum_{Daughters} \vec{F}_{d_i} + \sum_{Neighbors} \vec{F}_{n_j} + \sum_{Bonds} \vec{F}_{b_k} \tag{3.1}$$

If the point mass is moved, it changes the length of the `PhysicalCylinder`, and of its eventual daughters. This has to be recomputed with `updateLengthAndTension()`. And of course, the `SpatialOrganizerNode` might have to be displaced as well. Consequences of this last action are described in the next paragraph.

## 3.3 Designing modules

For this first example, we illustrate the programming interface of CX3D by implementing a previously published model of neurite outgrowth. In a series of papers, van Ooyen and his collaborators [48,99,100] presented a 2D model based on a production-consumption mechanism. In its simplest version, the soma secretes the intracellular substance 'tubulin' which diffuses along the neurite branches. The neurite distal segments (the growth cones) consume tubulin to move at a speed proportional to its concentration, and bifurcate with a constant probability.

To encode this simulation, we write three short java classes: two modules (a java class implementing the nine methods of the `LocalBiologyModule` interface, or extending the abstract class `AbstractLocalBiologyModule`) and one additional class to initialize the simulation.

Recall that each module is located within a `CellElement`. Instances of this first module will be located in a soma, where they secrete tubulin at a constant speed (the intracellular diffusion is processed automatically by the physics engine of CX3D):

```
public class InternalSecretor extends AbstractLocalBiologyModule {
    // secretion rate (quantity/time):
    private double secretionRate = 100;
    // (required by the super class):
    public AbstractLocalBiologyModule getCopy() {
        return new InternalSecretor();
    }
    // This is method is executed at each time step: secretion of tubulin
    // in the extracellular space with the modifyIntracellularQuantity
    // method of PhysicalObject.
    public void run() {
        super.cellElement.getPhysical().modifyIntracellularQuantity(
    "tubulin", secretionRate);
    }
}
```

The second module represents the growth cone. There is one instance of this class in each terminal

neurite compartment. It performs a smooth random walk (the direction is slightly perturbed after each step), with a speed depending on the concentration of tubulin, which is also consumed in proportion to the speed. In addition, the growth cones bifurcate occasionally, in which case copies of the module are inserted into the new daughter branches:

```
public static class GrowthCone extends AbstractLocalBiologyModule{
    // some parameters
    private static double speedFactor = 5000;
    private static double consumptionFactor = 100;
    private static double bifurcationProbability = 0.003;
    // direction at previous time step:
    private double[] previousDir;
    // the initial direction is parallel to the cylinder axis
    // therefore we override this method from the superclass:
    public void setCellElement(CellElement cellElement){
        super.cellElement = cellElement;
        this.previousDir = cellElement.getPhysical().getAxis();
    }
    // to ensure distribution in all terminal segments:
    public AbstractLocalBiologyModule getCopy() {return new GrowthCone();}
    public boolean isCopiedWhenNeuriteBranches() {return true;}
    public boolean isDeletedAfterNeuriteHasBifurcated() {return true;}

    // growth cone model
    public void run() {
        // getting the concentration and defining the speed
        PhysicalObject cyl = super.cellElement.getPhysical();
        double concentration = cyl.getIntracellularConcentration("tubulin");
        double speed = concentration*speedFactor;
        // movement and consumption
        double[] direction = Matrix.add(previousDir, Matrix.randomNoise(0.1,3));
        previousDir = Matrix.normalize(direction);
        cyl.movePointMass(speed, direction);
        cyl.modifyIntracellularQuantity("tubulin", -concentration*consumptionFactor);
        // test for bifurcation
        if(ECM.getRandomDouble()<bifurcationProbability)
            ((NeuriteElement)(super.cellElement)).bifurcate();
    }
}
```

Now we can set up and run the simulation, i.e. write a class to (1) define the substance 'tubulin'; (2) create a cell (quadruple `Cell-SomaElement-PhysicalSphere-SpaceNode`); (3) with an initial neurite segment; (4) place the local biology modules; and (5) start the scheduler:

```
public class ProductionConsuption{
    public static void main(String[] args) {
        // (1) properties of the intracellular substance
        double D = 1000;        // diffusion constant
        double d = 0.01;        // degradation constant
        IntracellularSubstance tubulin = new IntracellularSubstance("tubulin",D,d);
        tubulin.setVolumeDependant(false);
        //    registering the substance with the ECM class
        ECM.getInstance().addNewIntracellularSubstanceTemplate(tubulin);
        // (2) getting a cell (with the four abstract layers) at position (0,0,0)
        Cell c = CellFactory.getCellInstance(new double[] {0,0,0});
        // (3) create a neurite (pointing along the z-axis)
        NeuriteElement ne = soma.extendNewNeurite(new double[] {0,0,1});
        ne.getPhysical().setDiameter(1.0);
```

```
        // (4) insert production module in the cell's soma
        SomaElement soma = c.getSomaElement();
        soma.addLocalBiologyModule(new InternalSecretor());
        //    insert growth cone module into the neurite element
        ne.addLocalBiologyModule(new GrowthCone());
        // (5) run the simulation
        Scheduler.simulate();
    }
}
```

This model is extremely simplistic, but it already exhibits interesting properties: The elongation speed decreases with the number of terminal branches, but the bifurcation probability over time is constant, and so the distance between two branch points becomes shorter. In addition the tortuosity also increases as the speed decreases.

This example is well-suited to the CX3D framework because it relies on local computation by independent agents (in this particular case each growth cone's behavior depends exclusively on its intracellular concentration of T and MAP), and because it requires the modeling of physical processes (the intracellular diffusion). It also shows that the CX3D framework is general enough to allow the realization of other models from the literature.

## 3.4 Further examples

In this section we present three examples of different kinds of problems that could be simulated with CX3D. Each example was obtained by writing appropriate local biology modules and cell modules that provide the biological functionality required for each case. The first two examples are original models, and the final one is a previously published models now re-implemented in CX3D.

### 3.4.1 Layer specific dendritic growth

The first example (Figures 2.1, 3.5) illustrates the use of diffusible guidance molecules and how they can be used to produce layer-specific branching patterns of neurites [101, 102]. The simulation begins with an already-formed three layer cortex. The layers are formed of three different types of cells (L6, L5 and L4), all secreting a diffusible, layer-specific substance (for instance each L4 soma produces only the 'L4' substance, etc.). These substances diffuse through the environment, establishing chemical gradients that will guide the development of the axonal and dendritic neurites from two test cells inserted in L6, leading to a branching pattern that respects the layer specificity of pyramidal cells of layer 6. Namely, a down-going main axonal shaft, which produces side branches in L6 that move up to L4, where they

ramify, and an apical dendrite, also terminating in L4, but starting to branch earlier than the axons.

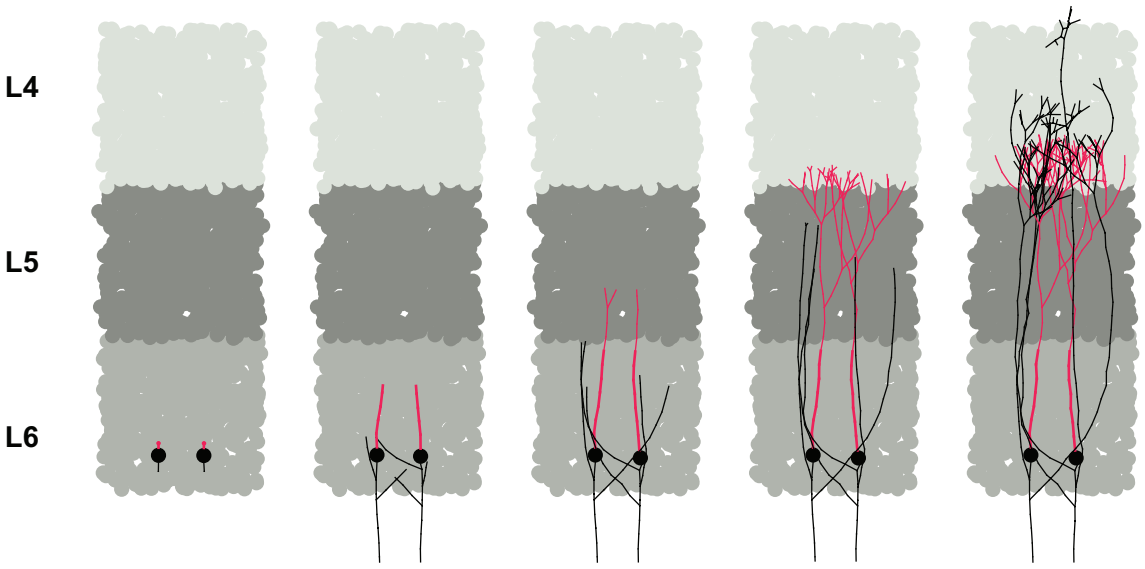


**Figure 3.5. Branching pattern based on extracellular signaling molecules**. In this simulation, we started with a column of a cortex-like tissue, with three layers composed of specific cell types (L6, L5 and L4, depicted in medium, dark and light grey respectively). Each cell secretes a layer-specific diffusible chemical, serving as guidance cue for the layer specific growth of the axonal and dendritic arbor of two test cells. The usual layer preference of typical L6 pyramidal branch could be reproduced: an apical dendrite (red) branching at the L5-L4 transition, and a down-going axon (black), with side branches in L6, moving up to ramify in L4.

In this simulation, each cell type forming the layers contains a single module, responsible for secreting the appropriate substance. The diffusion is performed automatically by the physics engine of CX3D. For the development of the branches, we wrote two small modules modeling the growth-cone functions and inserted them into the initial neurites. One of the modules elongates its neurite by moving its cylinder point mass either down the gradient of the L5 substance (for the axonal main trunk) or up the gradient of the L4 substance (for all other branches). The other module allows branching to occur with a probability that depends on the local concentration of a specific substance (L6 for the initial axon, L4 for the others branches). Different concentration thresholds for branching have been defined for the axons and the dendrites, and therefore the latter start their ramification earlier. Both modules are copied at branch points into the two new daughter growth cones. Neurite diameters decrease during elongation and at branch points, and the growth stops when the diameter has become smaller than a certain threshold.

The purpose of this simulation was not to reproduce exactly the morphological properties of specific cell types, but rather to illustrate the importance of long range inter-cellular communication through secretion and detection of diffusible cues.

### 3.4.2 Dissociated culture

Much can be learned from dissociated cell cultures, because their architecture is simpler than cells developing in-vivo, and because they are more accessible to imaging technics. Also, by growing cells on multiple electrode arrays it becomes possible to selectively record from, and to simulate, elements of a network. Some research groups have been interested in modeling this neuron-silicon interface, and have made growth simulations of neurons on a plate [103].

By restricting the cell movements to a very thin section of space, we can reproduce the 2.5D environment of cell cultures on a Petri dish. Our next simulation (Figure 3.6A and Video S4) shows twelve isolated cells on a plate, extending an axon and several dendrites. As in the previous example, each terminal neurite element contains a movement module and a branching module responsible for the extension of the cells. The main difference is that no guidance molecules are produced, so leading to the formation of an isotropic network.

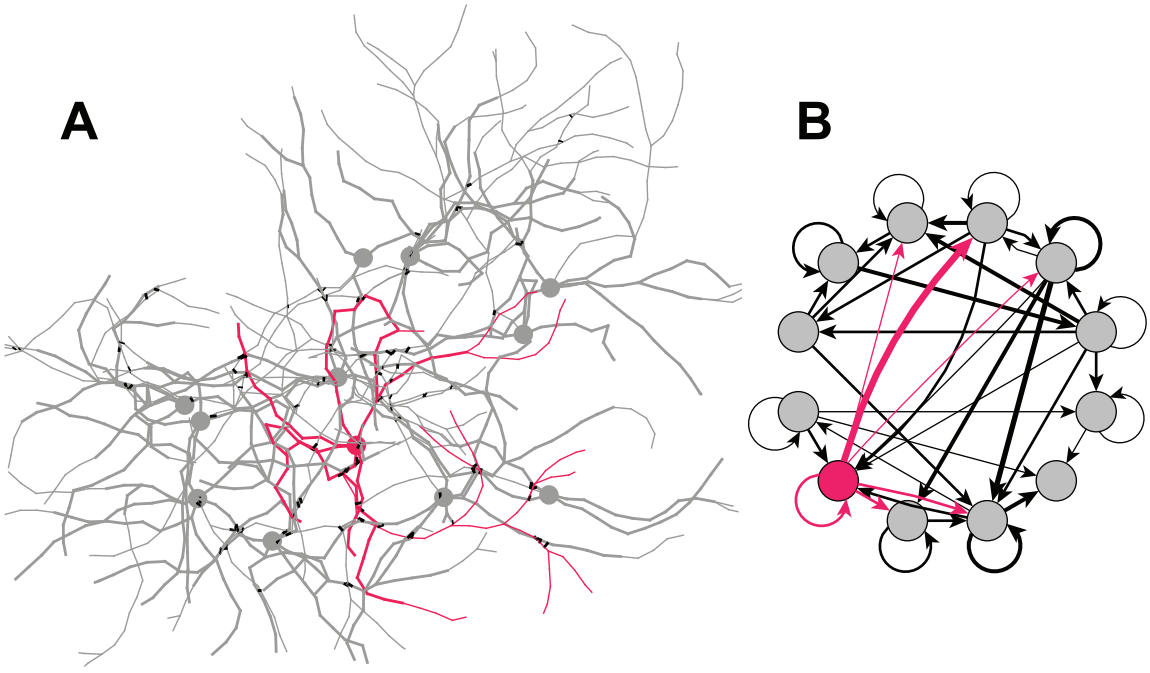


**Figure 3.6. Dissociated culture neurons forming a network. (A)** Eleven excitatory (grey) and one inhibitory (red) cells are randomly disposed in a 2.5D environment. They extend 4-7 neurites; one of them, thinner and growing faster, represents the axon. An attractive force between the cell elements induces a tendency to fasciculate. After the growth is completed, synaptic connections are formed randomly between neighboring axons and dendrites (black links). **(B)** Graph representation of the circuit shown on the left, drawn from a NeuroML description of the network, exported from CX3D after the simulation (black arrows: excitatory projections; red arrows: inhibitory projections; line thickness proportional to the number of synapses between cells).

This example illustrates two other features of CX3D. First, the possibility to change the cell-cell physical force. In this example, by increasing the range and the strength of attraction in the interaction between cylinders, we reproduce the fasciculation of neurites often observed in cultures. Secondly, the formation of a neuronal network: If an axonal neurite element comes into close contact with a dendritic neurite element, a synapse is formed with a certain probability between the two elements [104]. Neurons and their connections define a network (Figures 3.6B), whose description can be exported as an XML document that conforms to the NeuroML [105] description used for specifying electrophysiological simulations. This bridge from developmental to electrophysiological simulation offers a valuable tool for scientists interested in studying electrical activity in developing networks. Of course, it would be possible in future to extend CX3D to provide direct simulation of electrophysiology.

### 3.4.3 Contact inhibition

Lateral inhibition is an important mechanism for selecting – in an homogeneous population – individual cells that will adopt specific characteristics. One of the most studied pathway involves the transmembrane proteins Delta and Notch, from which Collier et al. [106] published a model: Notch is activated by the expression of Delta on the neighboring cells, whereas Delta is inhibited by the Notch level on the same cell. Additionally, both proteins are subject to exponential decay. This gives rise of a pattern of cells with a low Notch and high Delta profile, surrounded by cells with high Notch expression.

Collier et al. were mainly inspired by observations on Drosophila, but the Delta-Notch system is commonly found throughout neural systems development, including in the mammalian cortex where it is used to determinate which cells will acquire a neuronal or a glial fate. Therefore, we took it as a test example of how other models can be re-implemented in our framework. By doing so, the model originally developed on a 2D regular grid could be extended to a 3D agent-based version (Figure 3.7). In addition, now that it is coded in CX3D, it can be combined with other models in larger simulations. For instance to select the cells that will divide in a tissue.

Cell elements in CX3D can express membrane-bound substances. We designed a local biological module to regulate the expression for Delta (D) and Notch (N), according to the following dynamics:

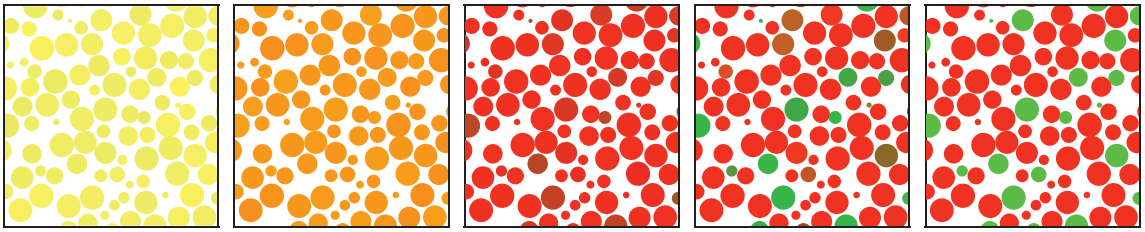

**Figure 3.7. Simulation of pattern formation by lateral inhibition (surface molecules)**. The simulation starts with a homogeneous population of cells expressing equal concentrations of the membrane bound ligand Delta (D) and its receptor Notch (N). According to the model of [106], each cell activates over time N in the neighboring cells, depending on its own D level, while decreasing its own D concentration based on its N level. The result is the selection by lateral inhibition of cells with a low N and high D. Such cells are not contiguous. (Red color intensity proportional to N, green proportional to D. Equal level of red and green intensity appears yellow).

$$\frac{dN}{dt} = f(\overline{D}) - N$$
$$\frac{dD}{dt} = g(N) - D$$

with $f(x) = \min(1, 20x)$, $g(x) = \max(0, 1 - x)$ and $\overline{D}$ is the average value of Delta on all the cells in close contact. This example is another illustration of the importance of modeling physics in a general purpose simulator (here to detect contact between close neighbors).

## 3.5 Performance testing

The execution speed of a CX3D simulation depends on the type of operations performed (in particular the proportion of physical objects that are moving). To test the performance of our framework, we present the CPU time required per time step for three different models. The simulation time step is $10^{-2}$ hour, and the speed of actively moving cell components is uniformly set at 100 $\mu m/h$. All tests were performed on a MacBook Pro with a 2.4 Ghz Intel Core 2 Duo processor, running Java 1.6.0.

The simulation of cell clustering shown in Figure 2.3G is close to the worst case scenario, with each single physical object moving at each time step, and each cell element containing a local biology module. For 2000 cells and 400 additional triangulation vertices (i.e. 2000 PhysicalObjects, LocalBiologyModules, CellElements and Cells, and two substances diffusing across 2400 extracellular volumes), the initialization,

i.e. the creation of all objects with the initial triangulation was done in 3.3 seconds, and the simulation of one time step took 400 ms. The images taken at 300 and 800 time steps are obtained respectively after two minutes and 5 minutes 20 seconds. It took much more time to have already formed clusters move and fusion into larger cells assemblies; the last image taken after 6000 time steps required 40 minutes of simulation.

The situation is much more favorable in our model of lateral inhibition, where objects don't move (and thus the triangulation is not modified), and where the substances are membrane-bound and thus don't diffuse in the extracellular space. For 2000 cells and no additional triangulation vertex, the simulation takes 63 ms per time step. The pattern presented in Figure 3.7 is complete after 400 time steps, i.e. 25 seconds.

Most simulations in practice correspond to intermediate cases, in which only a fraction of the physical objects are actively moving, as for instance in the model presented in Figure 3.5. For 1800 static somata and 100 additional triangulation vertices, with three extracellular substances diffusing, the simulation takes initially 135 ms per time step (at an early stage where the growing cells are composed of 140 non-terminal cylinders plus nine terminal cylinders with local biology modules in total). It requires 285 ms per time step at a later stage where there are 840 non-terminal cylinders plus 585 terminal cylinders. The total simulation time was 85 seconds.

## 3.6 Discussion

In CX3D, the users describe the simulation specifications by writing small mechanistic modules that are incorporated into the cells, defining the biological properties locally or at cell level. Using this approach, one can study growth and development in simulations of hundreds of cells.

In self-constructing systems, the environment (including physical laws) plays an active role in constraining the local interactions between agents. In the model presented in Figure 3.5, we could reproduce a layer-specific branching pattern, because the biological modules active in the terminal branches of the neurites could detect the diffusible signaling molecules produced by other cells. Associated with the possibility of expressing and detecting membrane substances, it offers the possibility to investigate by simulation a number of classical problems in developmental neuroscience, such as optical tectum map formation [107, 108], midline crossing [109], and, of course, more accurate models of cortical neuronal development.

These simulation methods demonstrate how morphology and function can arise out of implicit rules.

For instance in our second example (Figure 3.5), the desired shape of the adult neuron was not explicitly specified in the code. Instead, local decisions on whether to turn or to branch were taken independently in the growth cones, based on local chemical conditions, which lead to the final cell architecture. If the guidance cues had been secreted at different locations, or if they were absent, the resulting branching pattern would have been completely modified. Due to its modularity CX3D provides the ability to run the same biological models in different test environments, which is a valuable tool for a modeler interested in studying the relative importance of extrinsic and intrinsic factors. The model for a cortical cell can be tested in a cortex-like layered structure with several guidance cues, or in a sparse in vitro environment like the one of Figure 3.6. This kind of approach is interesting, in that it provides the modeler with two sets of constraints on a single set of parameters in the growth cone module. The fact that similar simulations can be run with various parameters, or after having selectively de-activated specific functions is also of interest for the study of mutations.

Our goal was to provide a general purpose simulation framework for the simulation of the physical development of neuronal networks. We showed how two models from the literature could easily be implemented in CX3D. Indeed, we could rely on the physics engine for technicalities like neighbor detection or diffusion, and did not have to code them anew. An obvious advantage in using the same framework for several types of simulations is that they can easily be combined in a larger simulation.

### 3.6.1 Future work

We have given several examples of how CX3D can be used to simulate growth of neurons in 3D space. Although we have the ability to generate synapses at contact points between neurons, these synapses are not functional, because our program does not yet incorporate electrophysiology. However, where the electrophysiology is requested, we provide the ability to export a description of a grown network as an XML document with the NeuroML level 3 specification [105] (http://neuroml.org). These documents can be used to configure a simulation in a point neuron electrophysiology simulator such as PCSIM [**?**] (http://www.igi.tugraz.at/pcsim). Future versions of CX3D could include an electrophysiology module directly inside neurites. Alternatively, modules could implement an interface for online communication with a coexisting electrophysiology simulator. This feature would of course be of great interest, because of the direct influence of electrical activity on neurite outgrowth [110], or on interneuron migration [111]; and in a later phase to study phenomena like synaptic competition [112] and learning [113]. A further limitation of the present version of CX3D is that it runs on a single processor, so limiting both the speed

and the size of simulations. However, we are currently developing a parallel implementation.

# Models

# Chapter 4

# Instruction language

## 4.1 Neural action primitives

To formalize the problem of the coding of sequential instructions for different types of neurons over time, we identified the minimal set of primitive actions that a neuron can perform during development. Our list contains the following elements: `move, secrete, detect, morph, fork, attach, replicate, synapse, die`. The expression of one of these primitives in a local region of a neuron provides it with a certain function. Arbitrarily complex sequences of actions can be described by composition of these low-level primitives. We also considered three additional primitives, `instantiate, kill, transport`, which are necessary for controlling the conditional expression of the primitive actions (Figure 4.1).

The expression of a primitive is always located in a specific discrete cell compartment, such as the soma or a neurite segment. This is in accordance with studies showing that different parts of a neuron can execute different tasks simultaneously (for instance an apical dendrite growth cone moving up the gradient of a signaling cue, while the axonal growth cone moves down the same gradient [114]), and – at least for a limited amount of time – independently (an isolated growth cone, cut from the soma, still extends its trailing neurite, and can even synthesize new proteins [115, 116]).

The following list presents the functional meaning of each primitive (for the detailed implementation in simulation, see the Methods section). All our primitives can be applied either to a soma, or to a neurite region (except for `replicate`).

1. **move** : This primitive regulates the active displacement initiated by the cell compartment, controlling both the velocity and the direction of movement. It represents all the machinery needed for the

displacement of a soma or of a growth cone: extension of filopodiae, adherence, traction etc. In case of terminal segments of neurites (growth cones), the forward movement of the segment is followed by an elongation of the neurite (new material is added), whereas a backward movement leads to shortening of the neurite (retraction). Non terminal compartments of neurite usually don't actively move.

2. **secrete** : Secretion of an extracellular (e) or intracellular (i) substance, or expression of a membrane molecule (m). The same primitive (with a negative sign) is used for active degradation/consumption. In order to describe the secretion of the extracellular substance 'X', one would write `secrete(X|e|+)`. If there is no risk of confusion, we might simply write `secrete(X)`. The location of this primitive in the cell defines the secretion location.

3. **detect** : Measures the concentration of a signaling molecule (extracellular, intracellular or membrane substance concentration). In case of extracellular substance, it also can detect a concentration gradient.

4. **attach** : Specifies the adherence properties of a cell compartment, by establishing mechanical links with other objects or with the extracellular matrix.

5. **morph** : Modifies the physical properties of a cell compartment, namely its diameter (both for somata and neurites), and its inner tension (for retraction or towed growth, for instance). This primitive is also used as a mecano-sensor, incorporating informations about the current physical conditions of the cell to the biochemical pathways.

6. **fork** : This primitive is used for making new neurite branches. In the soma it means the extension of a new neurite; In a neurite, it means the formation of either a side branch or a bifurcation (the later is only possible in distal compartments) .

7. **synapse** : Expression of pre- or post-synaptic structure, or establishment of a functional link between two such stuctures.

8. **replicate** : Represents cell division. This primitive is the only one which is restricted to the soma.

9. **die** : Elimination of a compartment, and all its dependencies. In the case of the soma, it stands for the death of the whole cell; in case of a neurite, it means death of a specific segment and all its distal branches.

The last three primitives don't represent neural functions or actions, but are being used to controle the expression or deactivation of primitives:

10. **instantiate** : Creation, from it's code description, of a *machine* (specific concatenation of primitives, see below) in a particular compartment. Its biological counterpart would be either the translation of a protein de novo, or the activation or liberation of a previously non functional protein (such as the fusion of a vesicle containing a membrane receptor, for instance).

11. **kill** : Permanent removal of a functioning machine. Represents either the degradation, or the inactivation of a protein (for instance the internalization of a membrane receptor).

12. **transport** : Shipping of a functioning machine into another compartment of the cell.

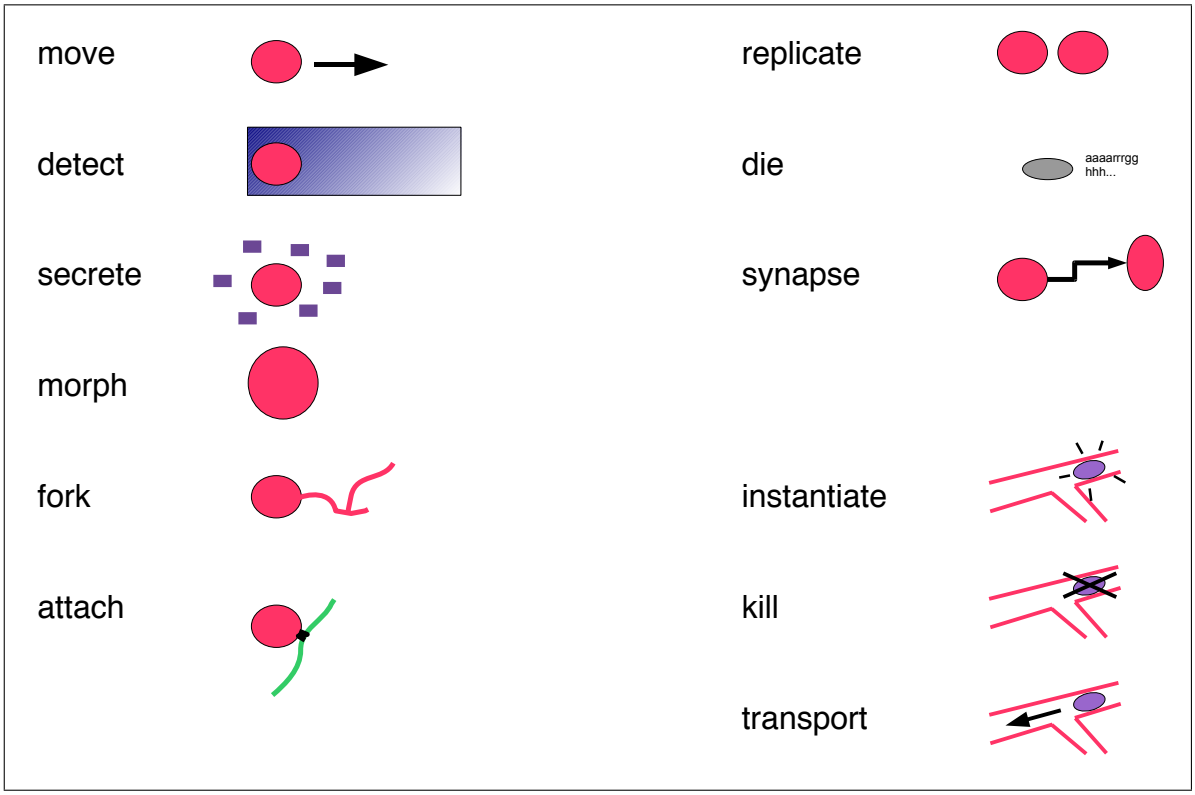


**Figure 4.1. The 12 Primitives of our instruction language**. Nine primitives represent *actions* taken by a discrete cell element; the three last primitives are used to *regulate* the expression of machines (concatenation of primitives) in a cell element

## 4.2 Combining primitives into neural growth motives

### 4.2.1 Machines

In metabolic pathways or transduction chains, different proteins can act on each other, performing modifications such as methylations or phosphorylations [117]. Biochemical networks thus define a directed graph, with the vertices being the proteins, and the edges representing the signal transmission. In axon guidance, for instance, when membrane receptors detect the presence of specific extracellular signaling molecules, they activate an intracellular signaling cascade, which reorganizes the cytoskeleton and results in a growth cone movement [118]. To express this pathway within our framework, we have the primitives `detect` and `move` at our disposal, which have to be linked in the correct order. In addition to representing an action, each of the primitives listed in the previous paragraph can be seen as a node in a biochemical pathway. This dual interpretation permits the genetic encoding of the developmental phenomena: A function is specified by listing the components (primitives and links) of the network realizing this function.

To assemble primitives into networks, we define *parameters* and *ports* (Table 4.1): A parameter, on the one hand, represents for instance the location and type of the signaling molecule that the detector can sense (such as in `detect(e|netrin)`, representing a receptor sensitive to the presence of netrin in the extracellular space). A port, on the other hand, is used for the communication between primitives, and specifies which aspect of the primitive is concerned. For instance, `move` has two input ports, 'direction' and 'speed'. Primitives are linked by joining one output port of an element to an input port of another element. For instance

`detect(e|netrin)`.gradient → direction.`move`

links the output 'gradient' of an extracellular detector to the input 'direction' or the movement mechanism.

In biochemical network, the signal propagation is often organized by scaffolding proteins, which can influence both the intensity and the duration of a signal [119, 120]. Therefore, in addition to primitives, our protein network can also contain filters (see Methods section for a list of filters):

`detect(e|netrin)`.gradient → `F` → direction.`move`

#### Formal definition

Biochemical pathways often contain recurrent modular patterns. In our framework, we will refer to such a module as a *machine*. It consists of a collection of primitives or filters (or even internal machines),

| primitive | input ports | output ports | parameters |
|---|---|---|---|
| `move` | speed<br>direction$^v$ | speed<br>direction$^v$ | $c_h, c_g, c_r, n$ |
| `secrete` | rate | | substance<br>(e\|i\|m)<br>(+\|−) |
| `detect` | | gradient$^v$<br>concentration | substance |
| `attach` | receptor_expression | adhesion_force | static/follow |
| `morph` | diameter<br>tension<br>cell_density<br>color | diameter<br>tension<br>cell_density<br>color | |
| `fork` | probability_to_fork<br>direction$^v$ | has_forked$^b$ | daughter diameter<br>machine(s) to instantiate |
| `synapse` | spines_formation<br>boutons_formation<br>connection_probability | connection_density | (excit\|inhib) |
| `replicate` | probability_to_replicate<br>direction$^v$ | has_replicated$^b$ | |
| `die` | probability | has_died$^b$ | |
| `instantiate` | probability_to_instantiate | has_instantiated$^b$ | machine to instantiate |
| `kill` | probability_to_kill | has_killed$^b$ | machine to kill |

**Table 4.1.** Primitives used in our instruction language, with their input and output ports, and their parameters. $v$ are ports with a vectorial value, which can be linked only to other vectorial ports. All other ports have a scalar value. $b$ are boolean i.e. either 0 or 1; they can be linked to normal scalar valued ports.

connected by links, with possibly inputs and outputs. A machine can have parameters, if the primitives it contains have parameters. Its formal definition is:

$$M = \{\mathcal{I}, \mathcal{O}, \mathcal{E}, \mathcal{L}\}, \tag{4.1}$$

where $\mathcal{I}$ is the set of input ports of $M$,

$\mathcal{O}$ is the set of ouput ports of $M$,

$\mathcal{E}$ is the set of machine elements (primitives, filters, conjunctors or machines) of $M$,

$\mathcal{L}$ is the set of links, namely valid ports pair (an input and an output port of elements of $\mathcal{E}$).

Our genetic code consists of the description, in a text file using the XML specification, of all the machines available. The description of a machine follows closely its formal definition, and contains a list of all elements, and a list of all links. Expressing a new machine means to create a machine following its

specification described in the code file. It is done with the primitive `instantiate`, which contains the address on the code (or the 'name' of the machine) where to find the description of the desired machine. Removing an existing machine can be done with the primitive `kill`. This mechanism implies that each machine is identifiable by a tag, so that only the desired one is destroyed. The biological justification is the local protein degradation control by the ubiquitin, which tags some proteins to be degraded. After destruction, a machine can always be re-instantiated later, since its description is still present in the code.

#### Complexity measure

This definition of machines also offers a way to quantify the complexity of the genetic code. We define the *complexity* of a machine is a function of the number of elements and links it contains:

$$C(M) = a|\mathcal{E}| + b|\mathcal{L}|, \tag{4.2}$$

where $a$ is the coefficient for elements and $b$ a coefficient for links (in principle we use $a = 1, b = 0.5$). When there is no risk of ambiguity, we will allow ourselves to describe a machine as $\mathtt{M} = \{e_i.\mathrm{o} \rightarrow \mathrm{i}.e_j\}$, where $e_i$ and $e_j$ are machine elements, and o and i output in input ports.

The next sections describe simple machines performing cell movements, differentiation and division. We then combine the code of such machines to form a larger instruction code, containing all the information needed for the generation of a multilayer neural network.

### 4.2.2 Movement

#### M1: basic chemotaxis

We now introduce the machine `M1`, the simplest machine for gradient ascent. This machine has no input, no output; it contains three elements: a `detect` primitive, a `move primitive`, and a filter (which defines if we move up or down a gradient). The machine also contains three links, to connect the 'gradient' output port of a `detect`, via a linear filter `LF`, to the 'direction' input port of a `move` (Figure 4.2). Its formal definition according to equation (4.1) is:

`M1(X,a)` = { {∅}, {∅}, {`detect(X),F(a),move`} { (`detect(X)`.gradient,
`F(a)`), ( `F(a)`, direction.`move`) } }.

`M1` contains two parameters, namely `X`, specifying the chemical that the detecting part is sensitive to, and `a` the constant of the linear filter that multiplies the gradient vector. If $\mathtt{a} > 0$, then `M1` represents the

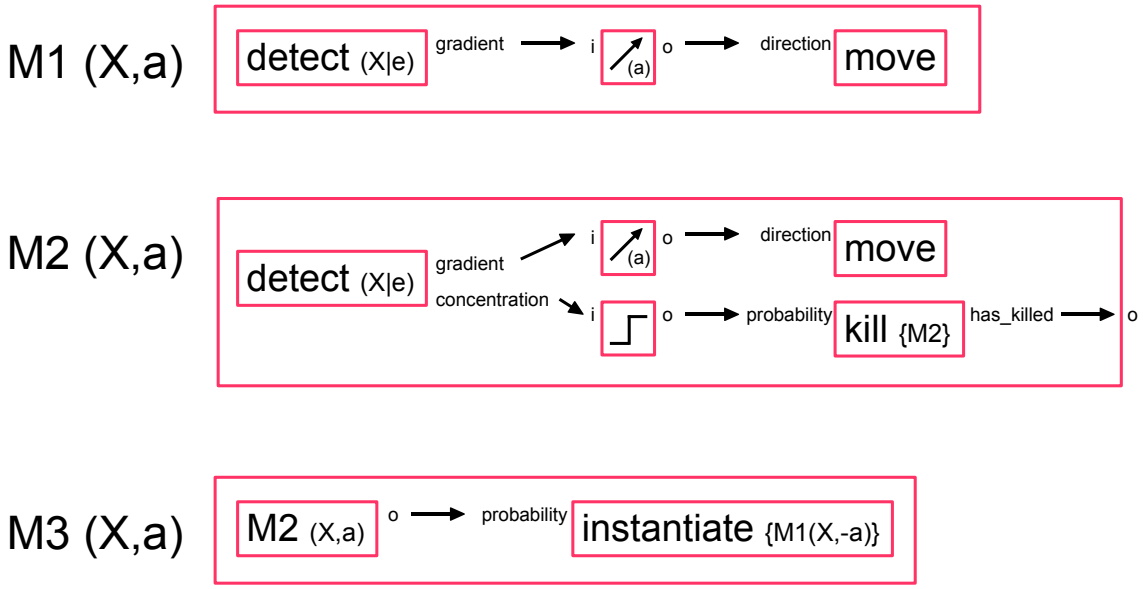


**Figure 4.2. Three simple machines, for movement guided by a signaling molecule X.** `M1(X,a)`: this machine contains three elements: a `detect` primitive probing the extracellular environment for the presence of substance `X`, a linear filter which multiplies a signal by the factor `a`, and a `move` primitive. It contains two links, connecting the output port `gradient` of the first element to the input port `direction` of the second element, via the filter (with input `i` and output `o`). The effect of this machine is to direct the movement of a cell element up the gradient of the signaling molecule X. `M2(X,a)`: this second machine contains the same elements than M1, with in addition a mechanism to kill the whole machine when a certain concentration of the chemical X is reached. Note the use of the second port of the primitive `detect`. We could write an equivalent machine using a M1 machine as element (see text). M2 has an output port, which transmits the information of whether it has been killed. This information is useful in `M3(X,a)`: this last machine initially contains an M2 instance, and thus moves up a gradient. When the internal M2 kills itself, it instantiates an M1 and moves down the gradient.

machinery needed to move up a concentration gradient; in the opposite case it represents the mechanism to avoid a substance. Figure 4.3A shows the behavior of a neurite containing an instance of `M1` in its terminal segment (the growth cone). The machine moves indefinitely, following the concentration gradient of a specified substance.

### M2: Movement and stop

Imagine we want the growth cone to approach an area with high concentration of a signaling cue, but stop at a certain point before reaching the peak concentration value (Figure 4.3B). A machine which executes this task could consists of the movement principle described in `M1`, combined with a mechanism to deactivate the machine at some point, for instance based on the chemical concentration.

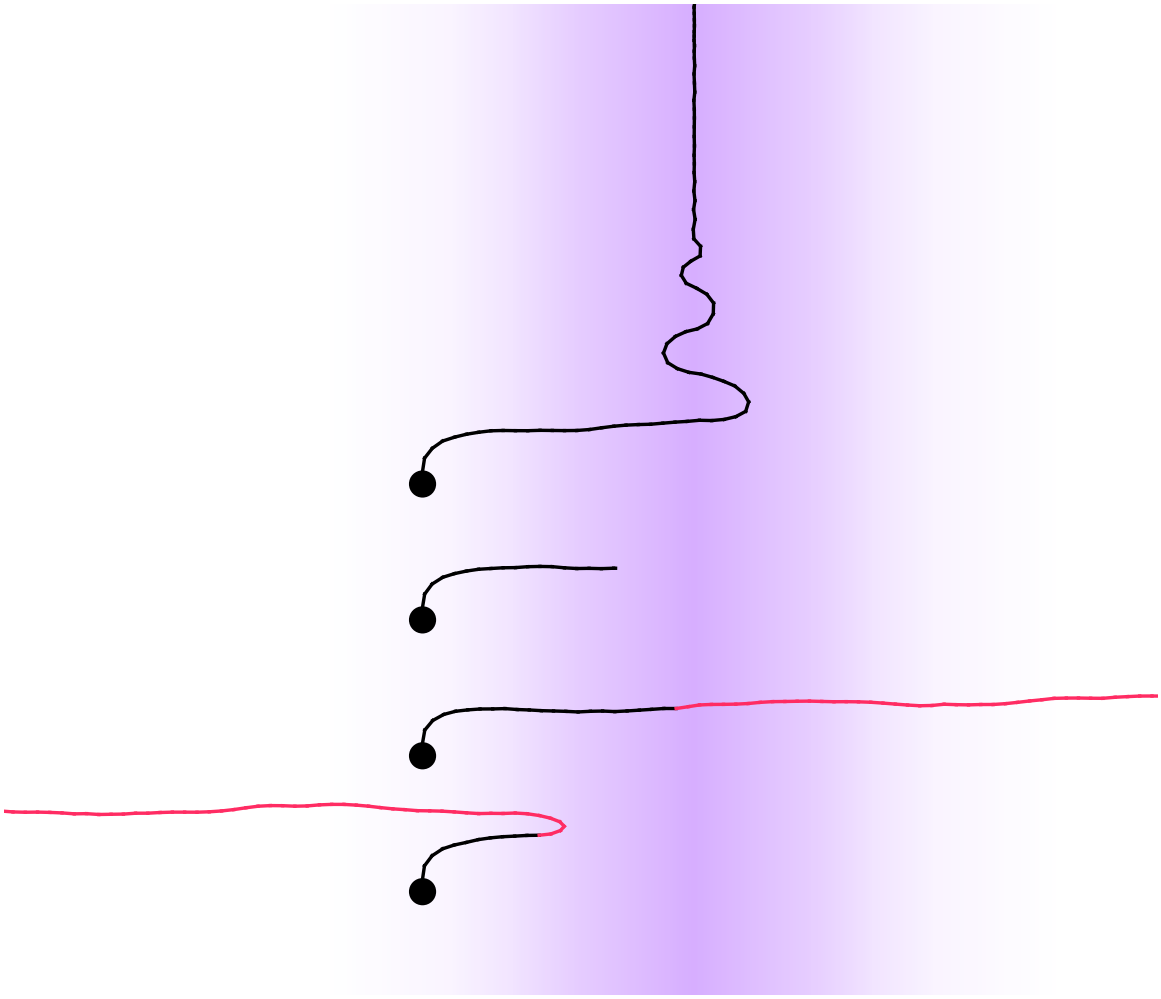

**Figure 4.3. The effect of M1-M3 in simulation**. Four cells are placed near a gaussian concentration profile (violet) of the chemical X. Each cell extends a single neurite containing one machine. The first cell contains M1, and therefore moves up the gradient, and stays in the plane (perpendicular to this page) where the concentration is the highest. The second contains M2 and therefore moves up the same gradient before stopping once it detects a certain concentration. The third contains M3 with an appropriate threshold. We see that it is first attracted by X, and then is repelled by it. The transition is shown by changing the neurite color. The fourth cell also contains M3, but the threshold for switching behavior is too low, and therefore it moves down the gradient before having reached the top.

$$\texttt{M2(X,a)} = \{\ \{\varnothing\}, \{o\}, \{\texttt{M1(X,a), detect(X), kill}\}$$
$$\{\ (\texttt{detect(X)}.\text{concentration}, \text{probability\_to\_kill}.\texttt{kill(}*\texttt{)}), (\texttt{kill(}*\texttt{)}.\text{has\_killed}, o)\ \}.$$

This formal definition of `M2` contains an instance of `M1` as a machine element, used for the movement part. This element is not connected to any other. In addition, the machine contains two other elements responsible for deactivating the machine: a `detect` and a `kill`. The 'concentration' port of the `detect`

is linked to the input port 'probability_to_kill' of the `kill` primitive instance.

Figure 4.2 shows an equivalent definition only based on primitives. Finally, note that `M2` has an output port *o*, which indicates when the machine has been killed. This next example shows how this output can be used for sequential operations.

**M3: Midline crossing**

Our third machine is a model of midline crossing, in which an axon is initially attracted by a signaling molecule, and then repelled by the same substance. The concentration of the signal is the trigger for the switch between these two behaviors:

`M3(X)` = { {∅},{∅}, {`M2(X,1), instantiate(M1(X,-1))`}

{(`M2`.*o* , probability_to_instantiate.`instantiate`) }.

This machine contains an `M2` instance with a positive filter value, and therefore initially the neurite moves toward the highest concentration of `X`. After it has reached the required concentration threshold, the initial `M2` removes itself. This removal activates the instantiation of a new `M1` instance with a negative filter value. This results in turning a chemoattractant signal into a chemorepellant signal. We see (Figure 4.3CD) that the threshold must be set precisely so that there is just enough momentum to cross the midline after the change of machine has occurred.

### 4.2.3 Branching patterns

The extension of a new neurite branch is controlled by the primitive `fork`, and results in the creation of a new autonomous cell compartment. This new compartment originally doesn't contain any machine, and therefore, for it to act as growth cone at least one new machine has to be instantiated. Usually, machine instantiation is done with the primitive `instantiate`. But since some machine instantiation systematically follows every branching, the forking primitive contains as argument the machine(s) to instantiate in the new branches that it creates, as if an implicit called to `instantiate` was systematically made. For side branch formation in a neurite, or for neurite extension in a soma, it takes only one set of arguments. In case of neurite bifurcation, there are two sets of arguments: one for each daughter branch. The two can be identical, for symmetrically behaving branches, or they can be different.

If a machine induces branches, and instantiates copies of itself into the new branches, we obtain a recursive branching mechanism. For instance :

`M4(X,Y,a)` = { `M1(X,a)` ; `detect(Y)`.concentration → probability.`fork` (`M4(X,Y)`)(`M4(X,Y)`)}

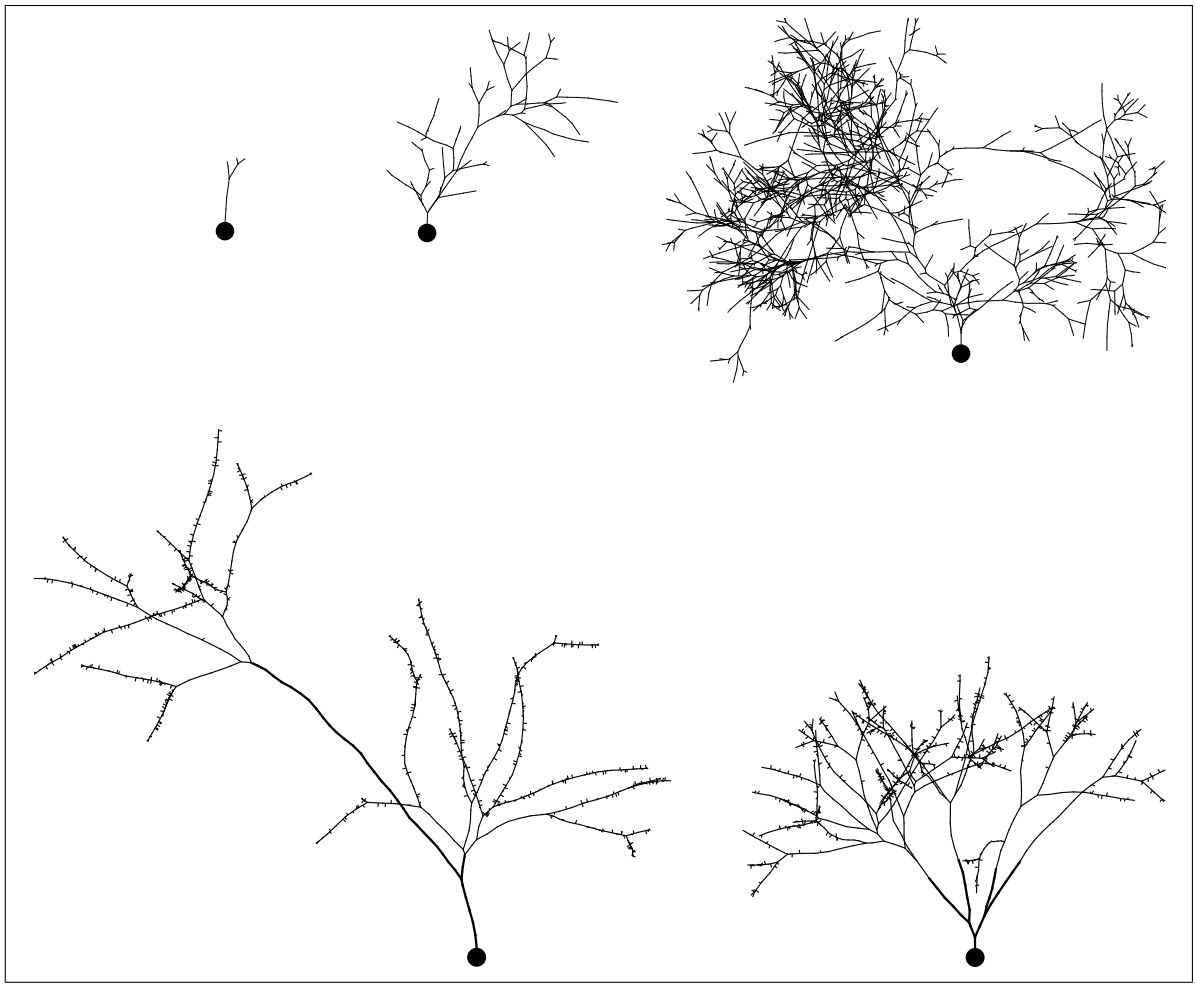

**Figure 4.4. Branching patterns with machines M5 and M7**. **(A)** Three different outcomes of the machine *M5* implementing a Galton-Watson process. The same probabilities (here $P_b = 0.01, P_s = 0.008$) can have very different outcomes: excessively small arbor, larger arbor or infinite branching (the simulation of the third run was stopped after 1200 time steps). **(B)** Outcome of the *M7* machine if the neurite diameter decreases at each bifurcation, with the elongation stopping once a minimal diameter is reached. Each terminal branch is at the same number of branch points from the soma, regardless of the actual length. **(C)** In the same machine, if the diameter is decreased only during elongation, each terminal branch is at the same length from the soma. Note the spines due to the machine M9 (see text), placed with a density inversely proportional to the neurite diameter.

The machine `M4` contains a movement component (a `M1` machine, described above), and a bifurcation mechanism, with a probability to bifurcate depending on the concentration of the signaling molecule `Y`. After bifurcation, a new `M4` instance is inserted into the daughter branches. If there is enough of the signaling molecule, bifurcation will occur repeatedly. To avoid infinite forking in this recurent system, it is necessary to include a stopping mechanism. We consider three different models:

**Galton-Watson model**

The first model that we can implement is the Galton-Watson branching process, which has already been used in the context of neural growth [121,122] : at each time step, each terminal branch has a probability $P_e$ to elongate, a probability $P_b$ to bifurcate and a probability $P_s$ to stop, with $P_e + P_b + P_s = 1$. This can be translated into our code:

`M5(b,s)` = { `M1(X,a)` ;

**1** → `F(b)` → probability.`fork` {`M5(b,s)`}{`M5(b,s)`} ;

**1** → `F(s)` → probability.`kill` {`M5`} }.

To obtain the correct probabilities to stop and bifurcate, we use the machine element **1** which always outputs one, and link it to a linear filter, the slope of which (denoted `b` and `s` in `M5`) defines the desired probabilities[1]. To ensure that the branching process will eventually stop, we need to have $P_b < P_s$ [123]. This condition comes with a very high rate of extremely short arbors (in more than 50% of the cases, the machine in the initial branch is killed before having bifurcated). Another characteristic of this model is that the same probabilities can have extremely different output, which doesn't make a pure Galton-Watson process a very reliable system for generating a desired branching pattern (see Figure 4.4 A). A solution is to have these probabilities changing over time, or based on some other conditions, like the concentration of some substance (see also next machine).

**Competition model**

The second mechanism to limit branching is the use of a production-consumption model proposed by vanOoyen [48, 99, 100]. An intracellular resource substance is produced at the soma at a constant rate, and diffuses intracellularly along neurites. The growth cones consume it for elongation: they move at a speed proportional to the concentration of the resource substance in the distal element, from which they remove a quantity proportional to how much they have actually elongated. The limited amount of resource limits the neurite outgrowth. The machine translation of such mechanism is as follows:

`M6_soma(Y)` = { `secrete(Y|i|+)` } in the soma, and

`M6_growth_cone(Y)` = { `detect(Y)`.concentration → speed.`move` ; `move`.speed → `secrete(Y|i|-)` } in the terminal elements of the neurites. It is of course also possible to have the branching probabilities depend on intracellular concentrations. Note that if the substance `Y` diffuses purely passively within the

[1] In contrary to a real Walton-Gatson process, where only one of the three options is chosen at each time step, the machine M5 would first check weather to bifurcate or not, and if not weather to kill the machine or not. To get exactly the same probability, we should use $s' = s/(1-b)$, which is almost $s$ if $b << 1$

neuron, there is no need to define a machine for its transport.

#### Diameter reduction model

The third branching process relies on a stopping probability depending on the neurite diameter, which can be reduced during elongation or at branch points [124–127]. The diameter is under the control of the primitive `morph`. This primitive contains one input port for gradual change in diameter, that we can use for slowly decreasing the diameter in proportion to the elongation, and one output assessing the current diameter, which can be used as stopping condition:

`M7(a)` = { `move`.speed → F(-a)→ diameter_change.`morph` ; `morph`.diameter → probability.`kill` }, with `a` being the diameter decay rate.[2]

The diameter reduction at branch points could also be coded with a specific machine. But it happens so often in our models, that we decided to have it as a parameter in the `fork` primitive. Let `b` be the value of this parameter. It is interesting to notice that different branching patterns arise from the choice of the two parameters used for describing diameter reduction during elongation and during branching. If `b` >> `a`, the number of branch points on the path connecting the soma to the extremities will be the same, regardless of the actual path length (Figure 4.4B). If `a` >> `b`, the path length from the soma to all distal extremities is identical, regardless of the number of branch points (Figure 4.4C).

### 4.2.4 Synapse formation

Synapse formation is an important part of neural development. Not only because it permits electrophysiological activity in the network, but also because there is a tight interplay between synaptogenesis and the formation of the axonal and dendritic trees. In our framework the primitive `synapse` regulates three different parameters: the production of a pre-synaptic terminal, the production of a post-synaptic density and the establishment of a functional synapse between existing pre- and post-synaptic structures. Each of these parameters corresponds to a specific input port. The output port of `synapse` indicates the density of connections made by a cell element. It can, for example, be used to prevent the retraction of branches that have already formed synapses.

We observe in pyramidal neurons that the bouton (pre-synaptic terminal) density is higher on terminal branches of the axon, which have a smaller diameter than on the main shafts. The implementation of such a mechanism is straightforward (Figure 4.4BC):

[2] `M7` causes a linear decay. It is possible to produce an exponential decay by modulating the decay rate withe the current diameter: { `move`.speed → ⊗ ; `morph`.diameter → ⊗ ; ⊗ → diameter_change.`morph`}, where ⊗ is a multiplicative node.

`M8` = { `morph`.diameter → `F` → bouton_formation.`synapse` }

where `F` is a linear filter accounting for an inverse correlation.

We can also propose more elaborated mechanisms. For instance, it has been observed that the binding of adherence molecules on axons and dendrites trigger the formation of pre- and post synaptic complexes [128]:

`M9_dendrite` = { `secrete(D|m|+)` ; `attach(A)` → spine_formation.`synapse` }

`M9_axon` = { `secrete(A|m|+)` ; `attach(D)` → bouton_formation.`synapse`}

where `D` and `A` are respectively a dendritic and an axonal membrane marker.

## 4.3 Differentiation and division

### 4.3.1 Gene regulatory network

Neural development requires a time- and cell- specific activation of genes to form different cell types, all containing the same genetic code[3]. Gene expression is under the control of transcription factors (TFs), small regulatory elements that bind onto specific regions of the genome (the promoter regions) and either activate or suppress the translation of specific genes. Being usually proteins, the concentrations of TFs are also regulated by gene expression. Several genes coding for TFs influencing each other's activity define a gene regulatory network (GRN). In addition, TFs also influence the transcription of another type of gene, coding for enzymes (performing a function in the cell) or for structural proteins (building bricks of the cell, like the cytoskeleton for instance). In the following, the genes whose activity is directly influenced by the GRN, without being part of the GRN themselves are termed *read-out genes*. In this section, we show how GRN genes and read-out genes can be coded with our primitives.

Several models of GRN have been proposed [129]. They usually consist of a set of differential equations describing how the activity $g_i$ of each gene (the concentration of the protein it is coding for) changes depending on the activity of all genes of the GRN. For instance through a weighted sum of all activities with a nonlinearity:

$$\dot{g}_i(t) = f\Big(\sum_j w_{ij} g_j(t)\Big) - k g_i(t), \tag{4.3}$$

where $g_i(t)$ is the activity of the $i$th gene, $f$ is a Hill function and $k$ a degradation constant [130].

In our framework, TFs are considered as simple substances (and not primitives). The left hand side of equation (4.3) describing the change of concentration, i.e. the production of the TF coded by the gene

[3]This section contains ideas developed with Sabina Pfister (see also Sections 4.4 and 5.2.1).

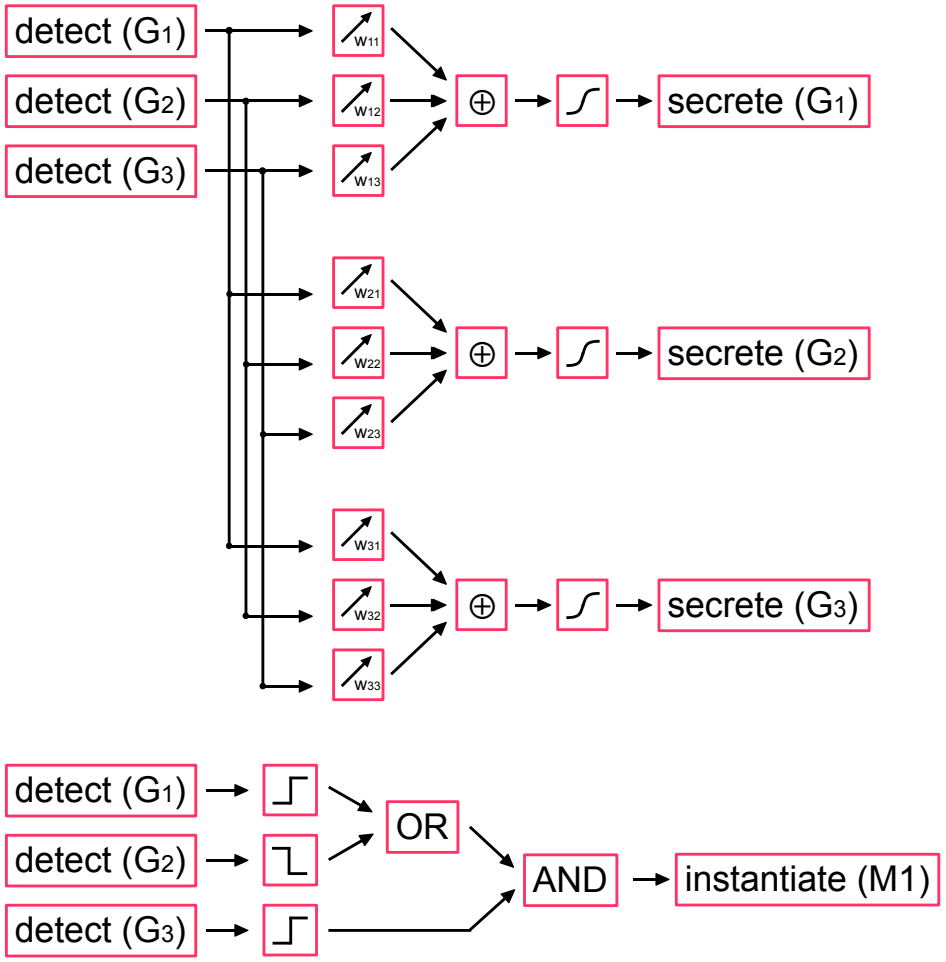


**Figure 4.5. A gene regulatory network (GRN) with a read-out gene**. (A) Several genes $G_i$ which influence each others expression define a gene regulatory network. Gene expression is done with the primitive `secrete`. The expression level is a function of the weighted sum of the concentrations of all genes in network (assessed with the primitive `detect`). The example represented in this figure consists of three genes, but it can be extended to an arbitrarily large number of genes. (Filters used: linear multiplication, sum, and Hill function). (B) A read-out genes instantiates a machine under some conditions on the GRN genes concentrations (Filters used: threshold functions, AND, OR).

$i$, is realized using the primitive `secrete`. The right hand side, defining the rate of production represents the effect of TFs on the promoter regions, is realized by a network of `detect` and filters (Figure 4.5 A).

We recall that the primitive `instantiate` constructs a machine from its code description. The read-out genes are thus composed of 1) a decision tree recognizing if the cell is in a proper state (i.e. whether the right level of TFs are present, assessed with `detect`) for triggering gene expression, and 2) an `instantiate` with the machine that has to be created (Figure 4.5 B). Note that, following the activation

of this gene, the newly constructed machine is located in the soma. It would have to be transported if it were to have a function somewhere along the dendritic or axonal arbors

It might seem odd that we use `secrete` for regulatory genes and `instantiate` for read-out genes. The justification is that an enzyme has to be present in a certain quantity in order to be able to execute its function. Its presence is thus considered as a boolean variable, defining if their is a high enough concentration to be active. Whereas the concentration of a TF is regarded as a continuous function. Moreover, TFs don't execute any action on their own. They just represent a signal that has to be detected for having an effect, pretty much like a signaling molecule. Models of boolean GRN have been proposed [131]. In such a model, the expression of a TF could also be represented by `instantiate`.

At cell division, the different TFs can be distributed asymmetrically to the two daughter cells, defining new (and possibly different) internal states. This asymmetrical repartition of gene activity permits the formation of several types of cells (see Methods).

### 4.3.2 Cell cycle

Another important regulatory network is the cell cycle, which controls the cellular division. It consists of several proteins acting as TFs or enzymes, activating and deactivating themselves in a complicated cascade, repeated for each division. Several mathematical models have been published (see [132] for review). Tyson proposed a particularly elegant model [133], describing the interactions between cyclin and cdc2 and exhibiting an oscillatory behavior. This model can be reduced to a system of two differential equations describing the evolution over time of two substances $u$ and $v$:

$$\begin{aligned} \dot{u} &= k_4(v-u)(\alpha+u^2) - k_6 u \\ \dot{v} &= \kappa - k_6 u \end{aligned} \tag{4.4}$$

with $k_4 = 100$, $k_6 = 0-5$, $\alpha = 1.8 \cdot 10^{-4}$ and $\kappa = 0.015$.

We implemented this dynamical system with our primitives: $u$ and $v$ are assessed with `detect`, whereas $\dot{u}$ and $\dot{v}$ are expressed with `secrete`, since the change of concentration depends on how much is secreted or removed. Note that this approach could be used to implement almost any kind of dynamical system $\dot{\mathbf{x}} = f(\mathbf{x})$.

We use the periodic oscillations of the $u$ substance of Tyson's model to define the different phases of the cell cycle (Figure 4.6). Since we don't model the physical replication of DNA, we only have two

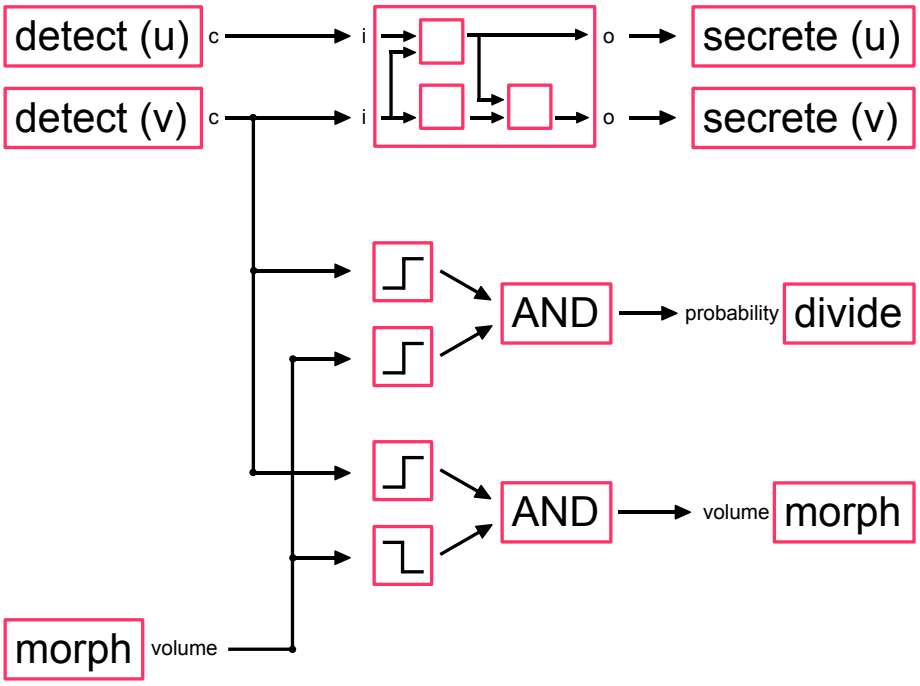


**Figure 4.6. Cell Cycle**. The concentration of the intracellular sustances $u$ ans $v$ are assessed with `detect`. According to these values, the machine produces or removes some of these substances, according to the model of Tyson. The concentration of $v$ is also a trigger for cell division (if the cell diameter is large enough). During phases of low $v$ concentration, the diameter of the cell is increased if it is small enough (such as after cell division for instance).

phases: the increase of volume (traditional $G1$ phase), which occurs at low $u$ concentration, and the division ($M$ phase) which requires both a high concentration of $u$ and a sufficient diameter. The second condition prevents multiple rapid divisions in forcing the cell to go through a low $u$ phase, where it can increase its volume, before being allowed to divide again (Figure 4.7).

## 4.4 Growing a three layered cortex

We can use the machines described above to grow a cortical structure composed of three layers (called L1, L2 and L3), with one cell type per layer. We want these cells to form the following network: L1 cells should project down to the basal dendrites of L2 and L3 cells. L2 cells should project to the basal dendrites of L3, and L3 cells should project up to the basal dendrites of L1 and to the apical dendrites of L2 and L3 (Figure 4.8A).

The code to generate this structure contains the description of 17 different machines (Table 4.2). This

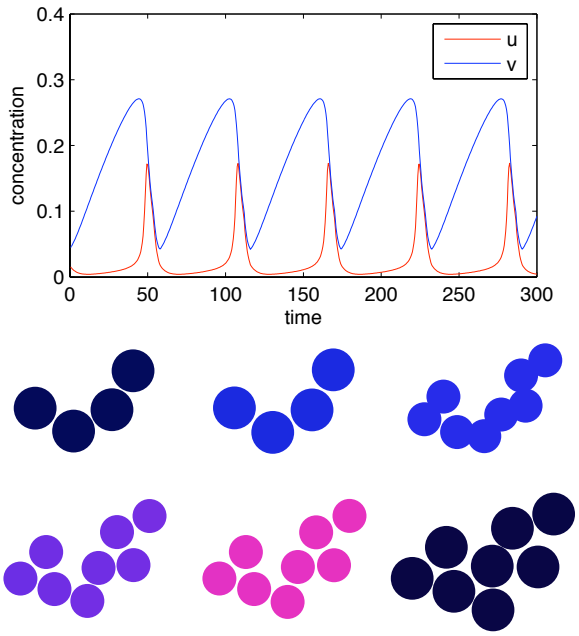


**Figure 4.7. Cell Cycle simulation**. (A) Evolution over time of the concentration of substances $u$ and $v$ measured during simulation in a cell containing a cell cycle machine. After each division, the measures are continued in one of the daughter cells. (B) Snapshots of the simulation, color coded with the red and blue intensity proportional to respectively the $u$ and $v$ concentration. Division occurs at the peak of $v$ concentration (blue), and results in cells with a smaller volume. Diameter increase occurs during phases of low $v$ concentration (black)

genome is inserted into a single progenitor cell, which immediately instantiates the first machine (`Start`). This machine instantiates a cell-cycle, a GRN, some read-out genes and an internal clock (constant production of an intracellular substance, the concentration of which defines a cell internal time).

The precursor starts to divide under the impulsion of the cell cycle. In the mean time, a GRN starts the expression of seven genes. The gene network used was designed by Sabina Pfister (and coded by Andreas Hauri) to produce sequentially cells with three different gene expression patterns (Figure 4.8B). Each of these patterns is recognized by the promoter of one of the three read-out genes, and leads to the instantiation of a differently parametrized `Migrate` machine, which moves the cell to a specific layer. In addition, the read-out gene machine kills the cell cycle, the GRN and all read-out genes, which stops the division of the cell and prevents the activation of another promoter. (Some basic principles of Pfister's GRN are presented in Section 5.2.1).

Once the neuron precursors have reached their final position, they wait for their internal clock to allow further differentiation into adult neurons: a cell type-specific forking machine is instantiated in the soma to produce the neurites, namely the basal dendrites (common to all three cell types), the main shaft of

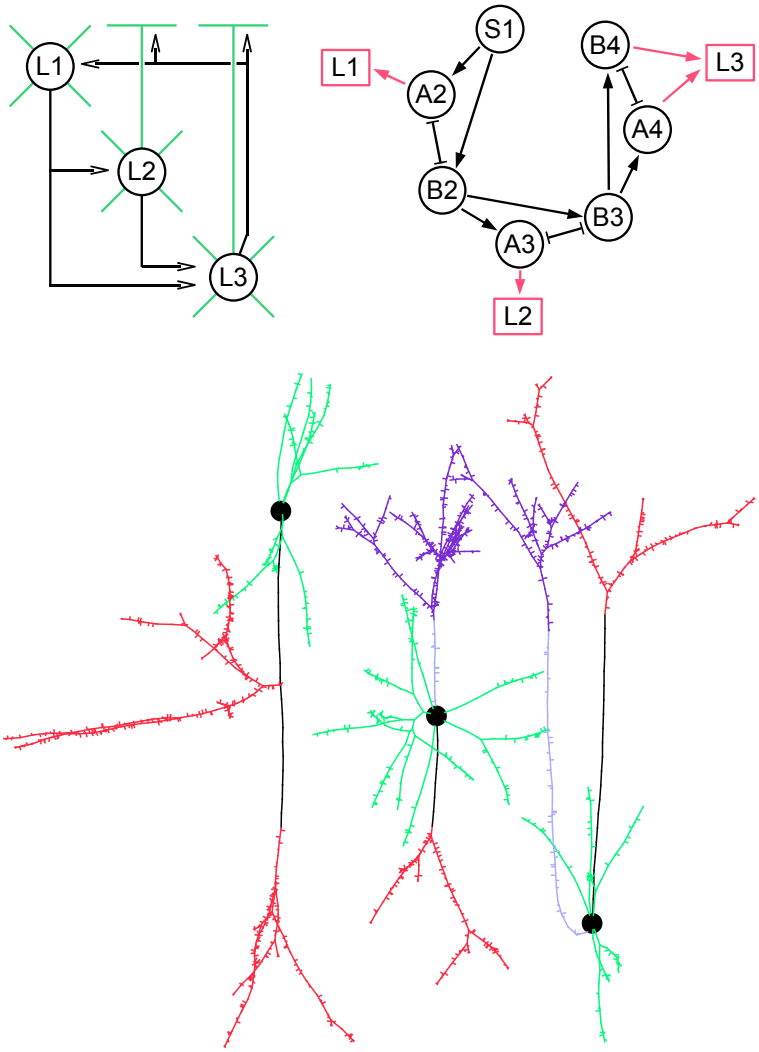


**Figure 4.8. Building blocks for the three layers cortex simulation**. (A) Schematic representation of the desired circuit. Axonal projections are depicted by a black arrow, apical and basal dendrites are depicted in green. Note that L1 cells have no apical dendrite. (B) Gene regulatory network (black) and read-out genes (red) for a sequential production of the three cell types. (C) Simulation snapshot of the three cell types. Identically colored segments are grown by instances of the same machine (black: axon shaft, red: axon ramification, light blue: apical dendrite shaft, violet: apical dendrite tuft, green: basal dendrite).

the apical dendrite (in L2 and L3 cells), and the main shaft of the axon (present in all cells, but with different parameters).

The terminal segment of each neurite now contains its own independent machine. The ones in the basal dendrites follow a random direction, and are rapidly killed with a probability depending on the (linearly reducing) diameter. The axonal and dendritic shaft move toward their target layer, guided by

the concentration gradient of the layer-specific signaling molecules. Once they enter a region where the concentration of the molecule they are sensitive to is high enough, the elongation machines kill themselves and instantiate a branching machine for the formation of an axonal or dendritic patch. The axonal shaft of the L1 cell also contains an additional machine that makes a side branch when the growth cone enters L2, and instantiates in it an axonal patch machine). The branching machines, and the dendrites shaft machine contain machines for the formation of boutons and spine. This example illustrates how the same code can be used to instantiate similar machines with different parameters (Figure 4.8C).

| id. | name | complexity (elements, links) | instantiates | kills |
|---|---|---|---|---|
| 1 | `Start` | 14.5 (14,1) | 2, 3, 4, 5 | |
| 2 | `CellCycle` | 55 (35,40) | | |
| 3 | `GRN` (7 genes) | 147 (87,80) | | |
| 4 | `read-outs` (3 genes) | 16.5 (12,9) | 6 | 2, 3, 4 |
| 5 | `Clock` | 4 (3,2) | | |
| 6 | `Migration` | 17 (13,8) | 7 or 8 or 9 | 6 |
| 7 | `L1SomaForking` | 9.5 (8,3) | 10, 13, 14 | 7 |
| 8 | `L2SomaForking` | 11.5 (8,7) | 10, 11, 13 | 8 |
| 9 | `L3SomaForking` | 11.5 (8,7) | 10, 11, 13 | 9 |
| 10 | `BasalDendrite` | 15 (11,8) | 16 | 10 |
| 11 | `ApicalDendriteShaft` | 8 (6,4) | 12, 16 | 11 |
| 12 | `ApicalDendriteTuft` | 14 (10,8) | 12, 16 | 12 |
| 13 | `AxonShaft` | 7 (5,4) | 15, 17 | 13 |
| 14 | `ApicalSideBranch` | 5.5 (4,3) | 15 | 14 |
| 15 | `AxonRamification` | 14 (10,8) | 15, 17 | 15 |
| 16 | `SpineFormation` | 4.5 (3,2) | | |
| 17 | `BoutonFormation` | 4.5 (3,2) | | |

**Table 4.2.** Machines contained in the genetic code used for the formation of the three layered cortex. The complexity is defined as the number of machine elements plus 0.5 times the number of links. For each machine, the third column lists the machine that it might instantiate, and the fourth column the ones that it might kill. For instance the machine `4` (the read-out genes), will instantiate the machine `6` (for migration) and kill the machines `2,3` and `4`.

## 4.5 Discussion and Conclusion

We have presented a way to encode developmental instructions for the growth and organization of neural networks. It is based on a set of primitives representing elementary neural functions. These primitives can be combined into large hierarchical networks that we called 'machines', similar to biochemical pathways, capable of reproducing complex developmental patterns. We have shown how a description of these machines can be encoded in an efficient way in a file that serves as genetic code for developing tissues.

### 4.5.1 The choice of primitives

In defining the list of primitives, our goal was to establish the smallest, complete, biologically plausible set. Meaning that (1) each element of the list is necessary, (2) this set allows us to reproduce every developmental step that a neuron can take during embryogenesis, and (3) each primitive should have a biological counterpart.

The first requirement is equivalent to say that none of the primitives can be expressed as a combination of the others. For some primitives, such as `secrete` and `detect`, it is obvious. But in some other cases, it can almost be a matter of taste. For instance, one could argue that `fork` and `replicate` are equivalent, the later being a fork at a cell level. We nevertheless consider them as separate elements because `fork` introduces a notion of hierarchy: a new neurite branch is subordinated to the cell element that executed the fork. Whereas `replicate` creates two daughter cells that are of equal level. Moreover, the mother cell that performed the replicate action doesn't exist anymore. Similarly we could object that `attach` can be written as a combination of `move` and `detect`. We have introduced this primitive because attachment can lead to passive movement, and because a single cell element can be attached to several other elements. Similar objections can be raised for the three regulatory primitives: is `transport` different than `kill` first and `instantiate` later in another cell segment? In our opinion it is, because such a cycle of degradation-creation would reset some internal states within the machine (such as the implementation at a later stage of a submachine).

The second requirement is even harder to prove formally. Our confidence in our primitives list stems from the fact that we haven't found yet any example of neural behavior occurring during development that would not be expressible within our framework. Of course we adopt a phenomenological approach to neural development. The set of primitives would change if we decided to increase the level of details at a molecular level, in which case we would have primitives describing the polymerization and depolymerization of microtubules for instance. Moreover, our primitives were only chosen to cover neural growth,

and the establishment of the first synaptic connections. For instance, a detailed implementation of the electrophysiological properties of the cells, allowing the description of different synaptic learning rules, would probably request new additional primitives. The focus on growth also makes our system inefficient for representing precise dynamical systems, as illustrated in our cell cycle model, which requires as much as 22 machine elements (primitives or filters) and 27 links to implement the Tyson differential equations (vs 13 machines and 13 links for the machinery doing the actual division and volume update based on concentration). Introducing the new primitive `solve`, which would take an equation, or a biochemical pathway - possibly expressed in a standard format like SBML (http://smbl.org) as parameter, would drastically reduce the size of the genome. This would allow the systems biology community to link their quantitative models of cell dynamic to the neural growth simulator CX3D. On the other hand, it is also an important feature of our coding scheme to be able to quantify how much information is needed to specify each machine. And if a sub-system can only be described with a complicated set of equations, its representation in the genome must reflect this condition. Finally, the current primitives don't allow the formation of syncytia, which would be needed to model a 'glial-syncytium' or even muscle fibers. A future version of this language applied to non-neuronal cells could incorporate the new primitives `fuse` and `split`.

The third requirement is the most crucial one, if one wants to be able to make any useful analysis or prediction from a formal approach[4]. `Move`, `secrete`, `detect`, `attach`, `synapse`, `replicate`, and `die` are obviously referring to well known biological processes. We are aware of the fact that each of these primitives describes in a single word extremely complex mechanisms, that might actually involve of hundreds of different proteins. But our intention here was not to model the detailed biophysics of the neurons, and therefore we allow ourselves a certain level of abstraction. The machinery needed for `morph` represents the interactions within a cell between the biochemical processes and the physical properties. The inputs of this primitive reflects the fact that a neuron can 'decide' on a biological basis to modify its cytoskeleton structure or its membrane material to change its volume or its tension. The output represents the influence of the mechanics on the biology which is also established: there are for instance receptors transducing the tension in a cell into biochemical activity, and the volume can be assessed by the variation of internal ionic concentration. `Fork` also refers to a well defined (although still imperfectly understood) biological process, usually involving a rearrangement -or loosing- of microtubules. The more

[4]Without this condition, we could push the logic to its limit and reduce the list to a single primitive (let's call it `X`). We define that `X-X` moves the cell element, `X-X-X` detects a signaling moldecule etc. It would be difficult to convince a biologist that he has to look for this omnipotent protein which, when combined with itself, is the only relevant actor in neural development!

arguable part is the fact that arbitrary machines can be immediately instantiated into the new neurites. There is no evidence so far for such a mechanism. Nevertheless, the numerous examples of stereotypical branching patterns (especially in insects), brings us to postulate that such a mechanism must exist. Similarly for `instantiate` and `kill`: there is a growing evidence that local mRNA translation plays an important role in neurite outgrowth, and that proteins can be temporarily or permanently deactivated. But it is not clear yet how many different sequential programs can be executed. Nevertheless, the example of midline crossing, and of guide posts shows us that it is possible to change the instruction within the same branch. It is usually admitted that protein synthesis requires a delay of several hours (ref.). We don't take this delay into account, but we could easily introduce it.

### 4.5.2 Complexity

In the three-layered cortex example, 60% of the genetic code is used for the generation of the three cell sub-populations (cell cycle: 15.3%, GRN: 41%, read-out genes with promoters: 4.6%); whereas the extension of the axonal and dendritic arbors only represent 40% of the genome (see Table 4.2). At first, this result might seem paradoxical: How can the amount of information required for producing these complicated geometrical structures be smaller than what is needed for merely producing three bunches of somata? The reason is that our genetic code doesn't specify each individual branch point. Instead, we encode an algorithm to grow the neurite arborization specifying the *probabilities* to bifurcate or elongate depending on the neurite's diameter or the concentration of a signaling molecule. A by-product of this encoding is that each instance of a neuron of the same type will be similar, but structurally different, which is also the case with cortical neurons. There are nevertheless cases in biology where the same exact branching pattern is found in every individual of a same species [134, 135]. In these cases it is probable that the mechanism leading to these stereotypical branches are specified precisely.

Decoding a neuronal shape, i.e. inspecting its geometrical properties to generate a construction code, requires the following steps: decompose the dendritic and axonal arbors in distinct regions with similar properties, and design a machine for growing each one of these regions. For example one machine to move straight, without branching to a certain layer, and then a machine to ramify there. This approach scales favorably if we increase the number of different cell types (for $n$ different types of cells of roughly the same complexity, the genome size needed would increase in the worst case in proportion to $n$. An increasing fraction of machines can probably be re-used in different cells). Note that this approach is different than grammar based rewriting systems. First because the correct unfolding of the program requires that the

proper signaling molecules have been expressed at the right location by the neighboring cells, and second because the cell morphology is actually grown, and not only statistically generated.

The problem is different for the GRN. The generation of the different cell types is also implicitly coded, since the population sizes are obviously not listed in the genome, but emerge from the interaction of the different genes of the GRN. But the interactions among these genes is specified explicitly. Which makes this process very expensive, at least in our formalism. One might argue that in biology these interactions are not explicitly coded, but that they arise from the physical and chemical interactions (such as the van der Waals forces) between the different transcription factors, regulatory mechanisms and translation machinery. Nevertheless these interactions depend on the molecular structure of the gene product, and so ultimately on the DNA sequence. The scaling is also defavorable: in the worst case, the amount of information needed to code the interactions between $m$ genes is $m^2$. If the number of genes influencing each gene remains constant, the complexity of the code grows lineraly proportional to $m$. It is possible for systems of $m$ genes to define $n$ different cell types with $m < n$. But this means diminishing the degrees of freedom when we design the system, and decompose an $n$-dimensional space (with a smaller $n$) in a larger number of domains $d_1, ..., d_m$ that have to be visited in a given sequence, if the order of cell type generation matters. Producing more cell types will always increase the problems of stability, of dependence on initial condition and influence of noise.

In summary, the developmental steps which need sequential, independent instructions are more stable and don't require much coding. Moreover, increasing the complexity is not detrimental to the stability of the rest of the system. For instance, having L1 axons making a side branche in layer 2 doesn't destabilizes any other features. Whereas events depending on the interaction of several sub-systems are costly in term of code size and robustness. For instance, adding another cell type makes the GRN extremely harder to define.

### 4.5.3 Toward a universal formalism for self-construction

We have demonstrated that our instruction code can be used to encode the developmental steps leading to the formation of neural architecture. The same primitives can be used to model the development of other organs or multicellular organisms, in which initial multipotent precursor cells divide, migrate, communicate, undergo apoptosis [136].

Would it be possible to describe higher-level biological self-organizing system with the same framework? Possibly the most famous example is the cooperative behavior in insects colonies [137]. Each single

individual behaves according to its local environment, which has been pattered by other members of the colony. The gender-specific actions like digging or take a piece of wood with the mandibles would require an additional primitives. But surprisingly many of the traditional insects behaviors can be coded with our primitives: produce pheromone (`secrete`), or follow a trail of pheromone (`detect` $\rightarrow$ `move`), lay eggs (`repplicate`) etc.

The computer science and robotics community has benefitted from the study of the principles of self-organization in biology, developing new technologies for distributed computing, modular robots or cell phone frequency allocation. The pursuit of this kind of studies will prove beneficial both to our understanding of biology, and to improve our future technological capabilities.

## 4.6 Methods

All simulations were conducted with the program CX3D described in the first part of this work. In CX3D, neurons are decomposed into discrete cell elements with distinct physical properties. These elements can contain *modules*, namely small programs written in java which describe the local biological properties of the cell element they are contained in. All the machines described in this paper are CX3D modules; They are thus associated with a particular cell element, either the soma or a neurite element. A machine contains machine elements and links, a machine element being a primitive, a filter or an other machine. At each time step (every $10^{-2}$ hours in simulation time), each machine element is run sequentially, and all inputs and outputs values are updated.

#### Primitives implementation

Most of the primitives are implemented in a straightforward way, using the standard CX3D API to either read information from the simulator's discrete elements, or to modify their physical properties. For instance `morph`, depending on its inputs or outputs, reads in cell element parameters like the diameter or the tension, or modifies them. `Fork` also uses the CX3D functions for creating new neurite branches, but after the branching this primitive instantiates a new machine, and inserts it into the newly formed cell element. `transport` is coded in a straightforward way: it removes a module from a cell element, and inserts it into the previous or the next element along the chain forming a neurite.

The `move` primitive is slightly more complex because it represents a model of the whole biochemical machinery used for the movement of a soma or of a growth cone. Several models have been published on chemoattraction and axon guidance (reviewed in Section 1.4). Inspired by Maskery and Shinbrot [49], we

consider that neural movements result both form deterministic and stochastic processes: At each time step we define a new displacement vector **d** depending on the previous movement direction **h**, the desired direction **g** (in case of chemo attraction) and some random noise **r** :

$$\mathbf{d} = c_h\mathbf{h} + c_g\mathbf{g} + c_r\mathbf{r}$$

where $c_h$, $c_g$ and $c_r$ are coefficients modulating the relative importance of the previous direction, the desired direction and the noise. These coefficients are parameters of the `move` primitive. The previous direction is the normalized sum of the previous movements, over the last $n$ time steps ($n$ is also a parameter for `move`)

$$\mathbf{h}(t) = \frac{\sum_{i=1}^{n} \mathbf{d}(t-i)}{||\sum_{i=1}^{n} \mathbf{d}(t-i)||}$$

Note that `move` represents only the displacement. The branching, which is often attributed to the growth cone, is done in our framework with the primitive `fork`. If we want the movement and branching mechanisms to influence each other, we have to explicitly code a machine link between these two primitives.

The `attach` primitive is conceptually simple but has the most complex implementation. Depending on the type of simulation, the 'static' adherence is implemented either by changing the inter-object physical properties (Figure 3.6A), or by adding supplementary springs joining two objects (Figure 2.6). The 'follow' port, used for fasciculation or migration along a fiber, is similar to the movement equation described above, with the desired direction **g** pointing toward a point distal on the guiding neurite.

#### Filters implementation

Apart from primitives, the nodes in a machine's network can be either other machines (which, as primitives, have input ports $x_i$ and output ports $y_i$), or filters. The list of filters that we use is:

Linear (or more precisely an affine transformation: $y = ax + b$)

Hill function ($y = x/(0.5 + x)$)

Inverse ($y = 1/x$)

Sum ($y = \sum x_i$)

Product ($y = \prod x_i$)

Step-function ($y = 1$ if $x > a$, otherwise $y = 0$ )

OR ($y = 1$ if at least one input $x_i \geqslant 1$, $y = 0$ otherwise)

AND (y=1 if all inputs $x_i \geqslant 1$, $y = 0$ otherwise)

These eight filters can be combined to form all other functions (for instance power two is done with a product filter with two times the same input.)

**Genetic code implementation**

All the biological properties of the simulated neurons derive from the machines currently expressed. The condition to express or remove machines is also specified in machines (containing the primitives `instantiate` or `fork`). The only exception to this rule is the *initial* machine, the one that has to be instantiated before the simulation can begin. To create a 'genetic code', it is necessary and sufficient to describe all the machines that can be expressed, and to specify which one is the initial machine.

Andreas Hauri [138] implemented an XML version of such a genetic code, where the machines are described according to their formal definition (4.1). The hierarchical structure of the XML specification is particularly adapted for listing for each machine the ports, the machine elements, and the links. The different parameters (like for instance for `secrete` the type of substance which has to be produced) are set using XML attributes. These attributes can be changed, so that the same machine can be re-used with different settings. The `instantiate` and the `fork` primitives contain as attributes the name of the machine they have to instantiate in the cell element, or in the newly formed cell element respectively. When these primitives are activated, they make a copy of the description of the machine that has to be instantiated, and in this copy they replace the parameters by the desired values. This new description is then parsed to instantiate the machine element, the links and the ports as Java objects. Finally, the machine is inserted into the CX3D cell element a cell element module.

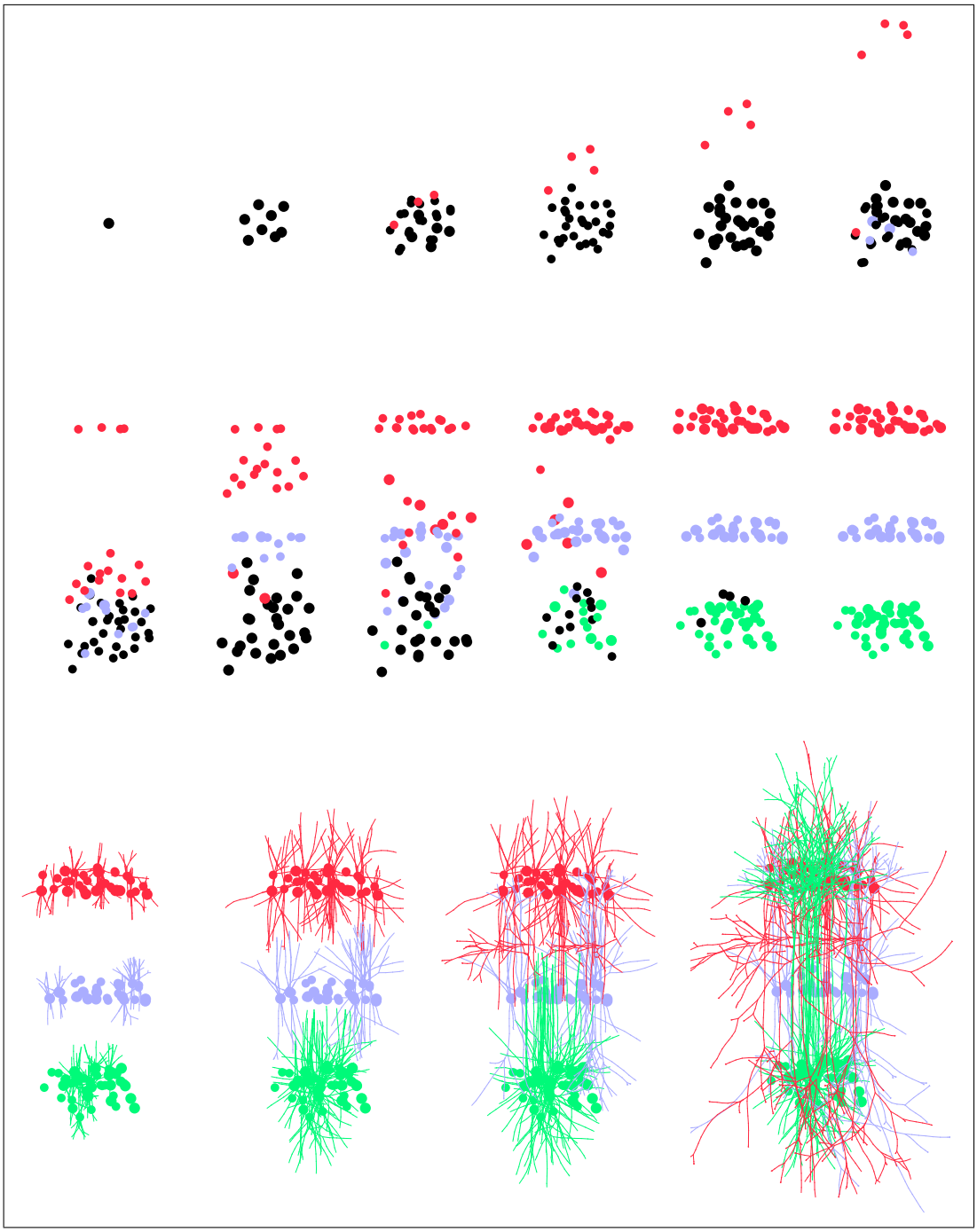

**Figure 4.9. Temporal sequences of the three layers cortex formation** A single initial precursor cell (black) divides, and form a precursor pool. Some cells differentiate into L1 cells (red), leaving the division cycle and migrating up the gradient of the 'L1' signaling substance. Progressively, L2 (blue) and L3 (green) cells are formed, which also settle to the regions with the highest concentration of their associated chemical. Finally, the neurons complete their differentiation by growing axonal and dendritic processes in a layer specific manner. In total, 89 cells are produced by the first precursor cell (for clarity, only one fourth of the cells have visible neurites).

# Chapter 5

# Toward realistic cortical simulations

The three layered cortex presented in Figure 4.9 suffers from two major over simplifications. First, the cells structures don't correspond to real biological morphologies. In this last Chapter, we use the same instruction language to reproduce the experimentally determined morphology of several cell types found in cat visual cortex. The cells that serve as model for this work were obtained from experiments on anesthetized cats by Prof. Kevan Martin and colleagues; the 3D reconstructions were made by John Anderson, and the integration into a convenient format for data manipulation by Tom Binzegger [139] and Prof. Rodney Douglas. In this work we consider four types of excitatory cells: pyramidal cells of layer 2/3 (P23), spiny stellate cells of layer 4 (SS4), pyramidal cells of layer 5 (P5), pyramidal cells of layer 6 (P6), and of one type of excitatory cells, the basket cells of layer 23 (B23).

The second limitation of the model described in the previous Chapter is the use of a pre-patterned environment (the chemicals used to define the position of the three layers are in place before the simulation begins). While it is true that biological systems use signaling mechanisms to configure and connect themselves, most of these signals are produced by cells within the system. In the model that is about to be described here, all signaling molecules are secreted by cells generated during simulation, in a truly self-organizing manner.

## 5.1 Preplate formation

As summarized in Section 1.5, future neurons are formed in the Ventricular Zone (VZ) and subventricular zone (SVZ), and migrate through the preplate (PP), composed of the subplate (SP) and of the future the layer 1 (L1) [also called the marginal zone (MZ)]. Therefore, to study the development of the cortex as

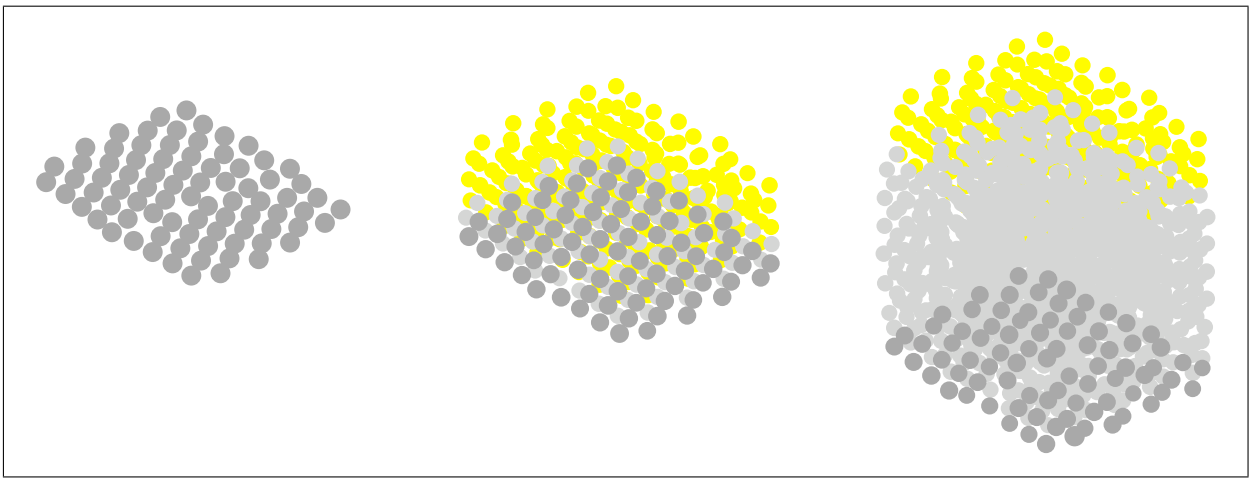

**Figure 5.1. Preplate formation**. Cells of the neural epithelium (dark grey) undergo asymmetrical divisions, forming first L1 cells (yellow) and then subplate cells (light grey). The cells at the bottom of the structure form the ventricular zone.

a self-construction mechanism, we have to simulate the PP formation before starting the corticogenesis. Unfortunately we have to be a little simplistic. First because the exact origin of several cells (like the Cajal-Retzius cells) is still an open question; And second because the neural plate comes from the ectoderm, which was formed during gastrulation [140]. Which means that to really start from a single cell in an exact sequence, we would have to model the previous developmental steps of the embryo, which is not the purpose of this work. Instead, we use our instruction language to create a three layered structure (L1-PP-VZ) without any external information, starting from one unique original cell.

In multicellular organisms, the initial cell (egg) has a polarity. The two 'poles' (called *animal* and *vegetal*) contain different concentrations of maternal proteins and mRNAs which are specifically partitioned during the first divisions [141]. Our simulation requires similar initial contitions. We suppose in our first cell, the existence of an North-South (NS) internal axis, as well as the presence of two intracellular substances `X` and `Y` in a precise concentration[1]. `X` is distributed symmetrically in every kind of divisions (whatever the axis of division is), while `Y` is distributed symmetrically when the axis of division is perpendicular to the NS axis and asymmetrically if the division axis is aligned with the NS axis, the daughter on the south side inheriting all of the substance.

The first steps in our simulation process will be to form an initial neural epithelium by a series of symmetrical divisions (with the division axis staying in one plane). This step doesn't require any external guidance, provided that the cells can use their internal polarity to organize the division axis. The number

[1] Another possibility would be to start with an existing machine secreting `X` and `Y`

of divisions can be controlled by the dilution of the intracellular substance `X` occurring at each division. As long as the concentration is above a certain threshold $\theta_1$, the cells divide in the plane perpendicular to their axis with a symmetrical distribution of the two inner substances. When the concentration of `X` is below $\theta_1$, the cells align the division axis with the NS axis, which results to an asymmetrical division: the `Y` substance is present only in one of the daughter cells (the 'southern' one). In our model, the presence of `Y` is necessary for the continuation of the cell cycle, therefore only one cell continues to divide. Even during the asymmetrical divisions, `X` is still distributed in both daughter cells. Its decreasing concentration defines the total number of divisions, as well as the type of the 'northern' daughter: either L1 or SP (Figure 5.1). After each division, the `X` and `Y` concentrations define the faith of the new cell by conditional expression of machines (Table 5.1). The initial `X` and `Y` concentrations, and the different threshold values $\theta_{1-3}$ specify the number of cells of each type that will be produced.

| Cell types | [X] | [Y] | Machines expressed |
|---|---|---|---|
| Horizontally dividing | $> \theta_1$ | | $\uparrow \rightarrow \perp \rightarrow$ `divide` |
| Vertically dividing | $< \theta_1, > \theta_3$ | $> 0$ | $\uparrow \rightarrow$ `divide` |
| L1 | $> \theta_2$ | 0 | `secrete(Reelin)`, `secrete(Semaphorin3A)` |
| SP | $< \theta_2$ | 0 | |
| VZ | $< \theta_3$ | $> 0$ | `instantiate(Cortical_Plate_GRN)` |

**Table 5.1. Preplate and ventricular zone formation.** The concentrations of the intracellular substances `X` and `Y` are used to define five different cell types, by the conditional expression of different machines. We don't show here the additional machines that control the volume and the color of each cell; they are similar for the five cell types ($\uparrow$ represents the internal cell axis, $\perp$ is a filter that accepts a $3D$ vector as input and outputs a perpendicular vector).

## 5.2 Cortical plate formation

After the preplate formation, the cells in the VZ start to divide and initiate the cortical plate formation, i.e. the establishment of a layered structure of neural precursors. This developmental phase can be subdivided into two parts: (1) the generation of the correct number of neuron precursors of each type and (2) their migration at the right position to form a layered structure.

[There are several modes of radial migration. In this work we only consider locomotion along radial glial processes (RGPs). RGPs are extensions of some of the progenitor cells attached to the pial surface, and are thus being elongated when the distance between the pial and the ventricular basal membrane

increases. While we see in principe how such a mechanism could be coded with our instruction language[2], the elongation of RGPs is not fully implemented yet. In the current version of the simulation, RGPs elongate in long fibers from some of the VZ cells just before the cortical plate formation starts. This problem will be addressed in future versions of our simulations.]

### 5.2.1 Cell lineage

The bottom-most cells of the ventricular zone divide to produce the different sub-populations of neurons. In our simulation, each cell with the appropriate internal conditions for being a VZ cell instantiates a gene regulatory network. As in Section 4.3.1, the GRN defines an inner state based on the expression profile of intracellular substances. The GRN used for this simulation was designed by Sabina Pfister [142]. It consists of 10 different genes influencing each other's expression rate according to the following dynamics:

$$\begin{aligned}
\dot{s}_1 &= 100h_1(s_1 + 1.0) - 66.6666s_1 \\
\dot{s}_2 &= 0.0638h_1(s_1 + s2) - 0.0387s2 \\
\dot{a}_3 &= 0.113h_4(1.3a_3 + 1.0s_1 - 1.0b_3) - 0.0992a_3 \\
\dot{b}_3 &= 0.113h_4(-1.0a_3 + 0.9s_1 1.3b_3) - 0.0992b_3 \\
\dot{a}_4 &= 0.113h_4(-1.0b_4 + 0.94b_3 + 1.3a_4) - 0.0992a_4 \\
\dot{b}_4 &= 0.113h_4(1.3b_4 + 0.9b_3 - 1.0a_4) - 0.0992b_4 \\
\dot{a}_5 &= 0.075h_4(-1.0b_5 + 1.3a_5 + 1.18\Theta) - 0.0658a_5 \\
\dot{b}_5 &= 0.075h_4(1.3b_5 + -1.0a_5 1.0\Theta) - 0.0658b_5 \\
\dot{a}_6 &= 0.075h_4(-1.0b_6 + 1.0b_5 + 1.3a_6) - 0.0658a_6 \\
\dot{b}_6 &= 0.075h_4(1.3b_6 + 0.8b_5 - 1.0a_6) - 0.0658b_6
\end{aligned}$$

with $h_1(x) = \frac{x}{1+x}$, $h_4(x) = \frac{x^4}{1+x^4}$, $\Theta = s_2$ if $s_2 > 1$ and 0 otherwise.

For cell division, we use a cell-cycle machine similar to the one discussed in Section 4.3.2. As long as it is not 'killed', it divides the cells at the desired rate, and distributes the intracellular substances (proteins) according to their 'asymmetry constant': The proteins $s_1$ and $s_2$ are distributed symmetrically during cell division. The other proteins are distributed asymmetrically: The first daughter cell receiving

[2]For instance `attach(basal_membrane)` for the adherence and `morph`.tension → length.`morph` for the elongation of the stretched fiber.

most of the $a_i$ proteins, while the second daughter cell inherits most of the $b_i$ proteins ($i = 4, 5, 6$). Some 'read-out' genes with specific promoters are then activated when certain condition on the GRN genes are met (Table 5.2)

| Cell types | Conditions |
|---|---|
| SVZ | $b_4 \geq 1$ |
| L6 | $a_3 \geq 1$ |
| L5 | $a_4 \geq 1$ |
| L4 | $(a_5 \geq 1)$ and $(b_4 \geq 1)$ |
| L23 | $\big((a_6 \geq 1)$ or $(b_6 \geq 1)\big)$ and $(b_4 \geq 1)$ |

**Table 5.2. Cortical plate formation**. The activity (i.e. the concentration of the protein product) of the genes $a_4, a_5, a_6, b_4, b_5, b_6$ define the conditional activation of cell type-specific machines.

Pfister designed the GRN to reproduce the cell lineage of cortical cells: Once the preplate is formed, cells in the VZ divide, increasing the pool of progenitors. The first decision point is under the control of the genes $a_3$ and $b_3$. These genes are auto inducing, and mutually inhibiting. The cells in which $a_3$ is expressed at a certain level quit the cell cycle and become L6 precursors. The other cells continue to divide, until they arrive at the second branch point, under control of the pair $a_4$ and $b_4$: individuals with high $a_4$ become L5 cells. The other form now the SVZ. Again, the same mechanism happens to produce sequentially L4 and L23 cells.

### 5.2.2 Migration

After the cell cycle exit, the newly formed neuron precursors perform a random walk. If they encounter a radial glial process, they attach themselves to it. The precursors migrate along the fibers, through the preplate, and stop before entering the marginal zone (MZ), the future layer 1. Their accumulation pushes the MZ further up. Next generations of neuron precursors migrate through their predecessors, leading to an inside-out development of the cortex: First L6, then L5, then L4, then L23.

We reproduce the initial random walk with a `move` without directional input. The adherence to the radial glial cell and the migration along the RGP is naturally coded with `attach(follow|RGP)`. Things are a little more complex for the stopping signal. Future neurons never enter the MZ. If the cells in the MZ express a signal (lets call it reelin), a simple stopping mechanism is: `detect(reelin)`→`kill(M)`, where `M` is the migration machine. This mechanism is sufficient if the cells are really produced sequentially

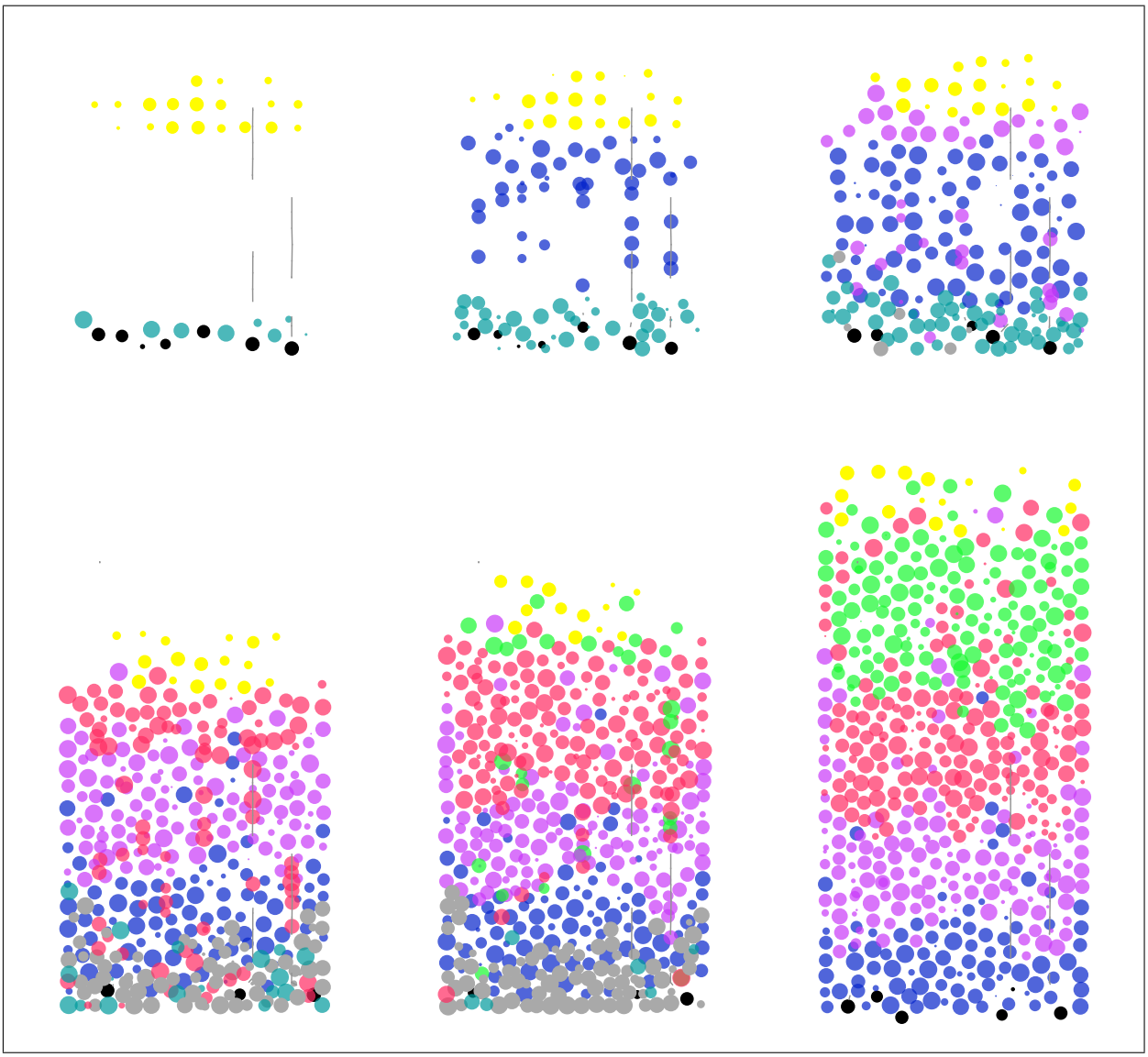

**Figure 5.2. Cortical plate formation**. Cut section through a cortical area during lamination. **(A)** After the preplate formation, the encephalon consists of the future L1 (yellow cells), the subplate (the cells of which have been remove in this simulation), and the VZ with some cells extending a radial glial process (turquoise and black cells). **(B)** The progenitor pool of the VZ increase, while L6 cells start to be produced (blue). The L6 cells migrate along the RGPs until they reach L1, or other L6 cells. **(C)** L5 cells are now formed; they migrate through the L5 cells (some of which are still being produced). The second pool of precursor is formed in the SVZ (grey). **(D-E)** L4 cells (red) and then L23 cells (green) are produced in the SVZ, and migrate toward L1. L1 cells are physically pushed upward. (F) Final laminated structure. Their is no VZ or SVZ anymore. The growth of axon and dendrites can start.

(first all the L6 cells, then all the L5 cells etc.). But if the GRN produces various cell types with a certain overlap (as is the case in biology), the last cells of layer $n$ are still being produced when the first cells of

the layer above $(n-1)$ are generated. We must thus incorporate an additional stopping mechanism which prevents a cell of layer $n$ to go through already settled cells of layer $n-1$. For that purpose, neurons in our model express a type-specific membrane cue; in addition, the cells which have already stopped to migrate express the membrane cue `s`. Our stopping mechanism kills the migration machine either if (1) the neuron precursor contacts a MZ cell, or (2) when the precursor is in contact with a sufficient number of cells of the same type that have already stopped their migration[3].

As in biology [143], several cells in our simulation end up in the wrong layer. Three mechanisms can contribute to migrational defects: (1) failure to stop, (2) stopping prematurely, (3) passive displacement after stopping. The relative importance of these effects depend more on the technical characteristics of the simulator than on conceptual issues. Changing the cell density, or the friction coefficient for non migrating cells, for instance, have a large impact on the performance of the layering. The border effects also play a disruptive role. To improve the layering quality, we decided to remove some of the misplaced cells with the following *apoptosis* mechanism: neurons that are in contact with less than three other cells of the same type are considered as being in a wrong layer, and eliminate themselves. This instruction can be coded within our coding language with a `detect` linked to a `die`.

A large part of the cortical inhibitory cells are produced in the lateral and medial eminence, and migrate tangentially into the cortex. Such cells are not produced by our GRN yet. As a proof of concept, we nevertheless incorporate one sort of inhibitory cells, which migrate laterally with a simple machine, following the gradient of a signaling molecule produced by L23 cells.

## 5.3 Axonal and dendritic branching

### 5.3.1 Geometry of cortical neurons

It has been known for a long time that neuronal projections in the cortex are not random, and that different subtypes of cells form specific patterns of axonal and dendritic branching, often in a layer-dependent manner [144, 145]. To be able to code a particular cell type with our instruction language and reproduce it in simulation, we use the following procedure: (1) we decompose the cell structure in smaller motives, (2) for each of these motives we design the machinery that would grow such a structure and (3) we link the machines in a logical order ("who instantiates whom", "who kills whom"). These principles

[3] And so younger cells settle at the bottom of their layer. We could also decide to stop the migrating cells when they detect a sufficient number of the cells of the *next* type; they would then stop at the top of their layer. The advantage of the current mechanism is that it is easier to code, since, for each cell, it only contains the reference to its own membrane marker.

were applied to five different cell types. We illustrate the method in detail with the pyramidal neurons of layer 2/3 (P23) and pyramidal neurons of layer 6 (P6).

As was already described in [101,139,146], we see in Figure 5.3 that P23 have their main axonal shaft going down toward the white matter, forming tangential collaterals in L2/3 and in L5, but not in L4 or L6; whereas P6 emits branches in L6 which move up to L4 to ramify (neither the main axonal shaft nor the collaterals does branch in L5). These differences are due to the layer-specific expression of signaling molecules, such as EphrinA5 which is present in L4 and in L6 and acts as repulsive signal for P2/3 axons and branch promoting signal for P6 [147].

### P23 code

As for the cells of our three layered cortex (Section 4.4), the first machine used for neural differentiation extends the neurites from the soma, in the appropriate direction. P23 neurites are oriented in specific directions (Figure 5.4A): The axon leaves the soma in the 'inferior' pole and goes 'down' toward the white matter. The apical dendrite extends from the 'upper' pole, and moves 'up' to L1. In biology, a gradient of Semaphorin3A spans the whole thickness of the cortex, and guides the movement of the axon and of the apical dendrite [114]. In our model, we use the same signaling cue to direct also the original extension of the neurites from the soma[4].

Figure 5.4B shows the initial machine used to construct a P23 cell: it simply extends a neurite in direction of the Semaphorin gradient, one neurite in the opposite direction and a small number (2-7) of neurites in random directions in the plane perpendicular to the semaphorin gradient. In these neurites, the initial p23 machine instantiates the machines to respectively start a P23 apical dendrite, a P23 axon or a P23 basal dendrite.

The P23 axonal growth cone goes down the gradient of Semaphorin3A, which is realized as usual with a `detect` and a `move`. In addition, while in layer 23 or in layer 5, the axon can extend side branches, a function for which we use a combination of `detect` and `fork` (Figure 5.4 CD). These side branches in L23 or L5 elongate (staying in the same layer), and after a while ramify. To code this behavior we use the mechanism described in Section 4.3.4 : The extension decreases the diameter of the neurite, and the diameter influences the probability to bifurcate (with the insertion of similar machines into the daughter neurites) or to stop, by simply killing the machine (Figure 5.4 E).

[4]Actually, this polarization of the pyramidal neurons already occurs before the migration: the leading and trailing processes of the migrating neuron precursor developing into the apical dendrite and axon receptively [79]. However, the principle is the same: at one point the neuron has to define an axonal and a dendritic pole.

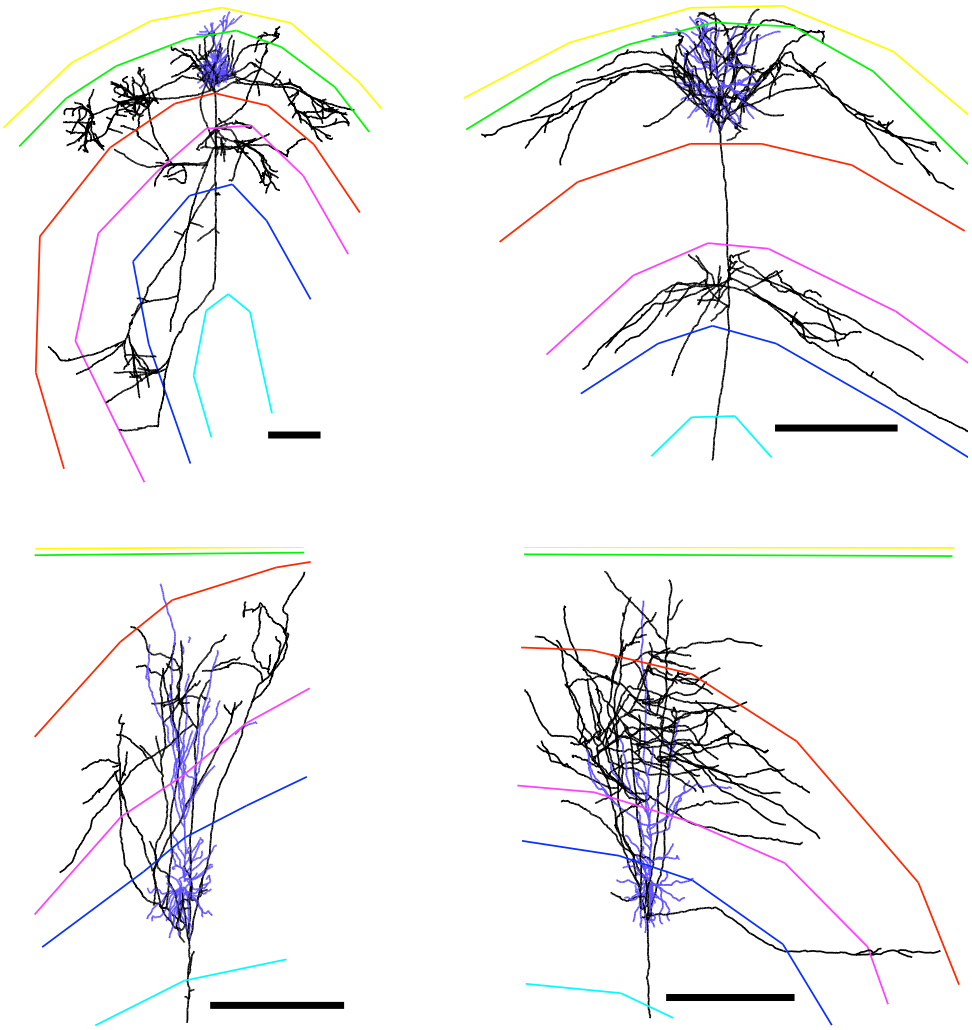

**Figure 5.3. Pyramidal cells of layer 2/3 and layer 6 from the cat visual cortex. (Top)** P23 cells have their soma in L2/3. The axon's main shaft goes down to the white matter, and extends collaterals in L2/3 and in L6. These collaterals tend to stay into their layers, where they can ramify. **(Bottom)** P6s main axonal trunk also goes down; they extend side branches in L6, which move up to ramify in L4. There is very little branching in L5. (Scale bar: 100 $\mu m$. Cells reconstructed by John Anderson, INI).

### P6 code

The initial P6 machine is very similar to its P23 counterpart: it produces an apical dendrite, an axon and several basal dendrite. The only differences are the type machines instantiated into the new neurites (Figure 5.5 AB). The initial axonal growth cone is also very similar to the one in P23 : it moves down the semaphorin gradient, and produces collaterals while in layer 6 (Figure 5.5 CD). Here also, the difference with P23 is the type of machine instantiated in the side branches: The P6 first collaterals move up

toward layer 23 (Figure 5.5 D). Once in that layer, they make another type of side branches, which move tangentially inside layer 23, and finally ramify. For simplicity we use the same procedure than for the P23 side branches (Figure 5.5 E). Interestingly, although P23 and P6 cells seem first to be very different, they share most of their machines (with different parameters, namely the identifier of the next machine to be instantiated).

#### Three other cells

We apply the same principles to three other cell types (Figure 5.6): The basket cell of layer 23 (B23) has a down going axon which can bifurcate and produce two types of collaterals. The proximal collaterals are formed close to the soma and surround it with branches. The distal collaterals are much shorter and branch much less often. The spiny stellate cell of layer 4 (SS4) axon has first-order side-branches which travel horizontally; these fibers have then second-order collaterals which move up and ramify when they contact layer 23. Finally, the main trunk of the axon of the pyramidal cells of layer 5 (P5) also has two sorts of collaterals (similarly as the B23): the first ones create a patch next to the soma, whereas the second ones move and ramify in layer 23.

## 5.4 Conclusion

Modelers are always being asked to make predictions. In Chapter 4, we have seen that the information needed to code the cell lineage of a small network is larger than the information needed to specify branching patterns. Because this statement makes sense in the context of the Kolmogorov complexity of these different tasks, we imagine that this also holds for more complex networks, such as the neocortex. This result can be considered as a prediction that we formulate. But predictions are probably not the most useful outcome of our simulations.

During the design of both the simulator and the instruction code, we often had to make assumptions or introduce new concepts. On several occasions, we discovered later that these postulated concepts corresponded to well known biological facts. We can cite for instance the role of inner tension in a neurite metabolic extension, or the scaffolding proteins maintaining in physical proximity different proteins of a biochemical network, and modifying the signal transduction between them. On many occasions, we couldn't find in the neuroscience literature a satisfying answer to our questions.

These questions should be of interest for the biologist working on neural development. Because they point to flaws in our current knowledge. But also because they are so crucial, that we, as modelers,

couldn't avoid dealing with them:

**1)** What are the stopping conditions for radial migration along the glial fibers. By this question we mean not only: how do cells not enter the MZ, but also: how does a late-born cell of an early type know that it has to stop in the correct layer? This is the prototypical example of a problem for which we *had* to invent a solution. Without stopping mechanism, there is no self-organizing model of cortical development. Note that this problem is the subject of active research.

**2)** How are sequential instructions coded? For instance during pathfinding, an axon going from point A to point D often use many intermediate goals, like guide post cells, going step by step from A to B, than to C, and then finally to D. Are all these instructions already present in the initial growth cone, or are they shipped at a later stage. This problem has be studied in the context of midline crossing for instance.

**3)** How can different daughter branches at a bifurcation adopt different behaviors? Is this already determined at the start of the initial growth cone?

**4)** How is the desired length of a neurite specified? For instance how come that all the basal dendrites of a pyramidal cell have very similar length, regardless of the number of branch points they contain?

**5)** How is the number and initial directions of collaterals specified? All pyramidal cells of a certain type have the same order of magnitude of side branches leaving the main axonal shaft. And we see that they don't leave the trunk in random direction, otherwise there would be a large number of neurons with all branches on the same side, like in our model of a P5 cells (Figure 5.7). This is an example of key question for us (because we have to program such mechanism in our simulation), that is not part of the current main stream of biological research in development.

**8)** Is branching by bifurcation of the growth cone different from the extension of a side branch on an already existing shaft?

**9)** There are two ways to look at a branching process. Either as a binary tree: one branch of order 1, two branches of order 2, four branches of order 3 etc.; or like a hierarchical tree: one branch of order 1 with several collaterals (order 2), which in turn have collaterals (order 3) etc. Which model is more accurate in neurons? Is this different for axons and dendrites?

The list is not exhaustive. And many of these problems are already recognized as major problems in neuroscience, and the subject of active research. Our opinion is that they play, maybe bore than

other more popular problems, a very important role in the computational characterization of neural development.

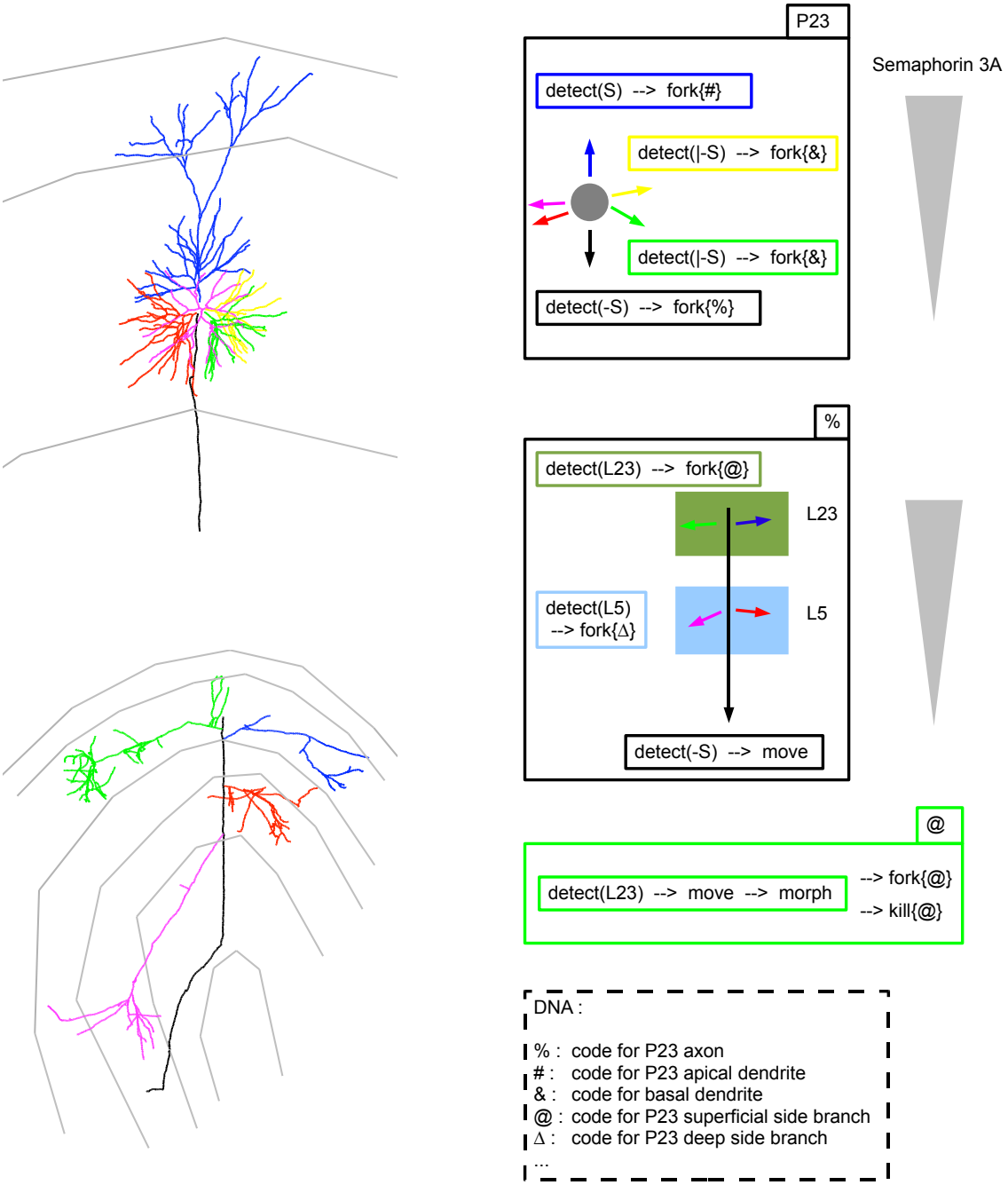


**Figure 5.4. Motives of P23 neurons**. **(A)** Neurites of a P23 neuron: the apical dendrite (blue) is the prolongation of the soma in the pial direction, and extends toward L1. The axon (black) leaves the soma in the opposite direction. The basal dendrites are extended tangentially. **(B)** The initial machine used for the neurite extension, first step in neuronal differentiation. Its correct function depends on the existence of a concentration gradient of the substance Semaphorin3A). **(C)** The main trunk of the axon goes toward the white matter, and some of the collaterals produced in L23 and L5. **(D)** The P23 axon main trunk machine. **(D)** The L23 collateral machine. **(F)** The cell's DNA contains a description of all the machines used.

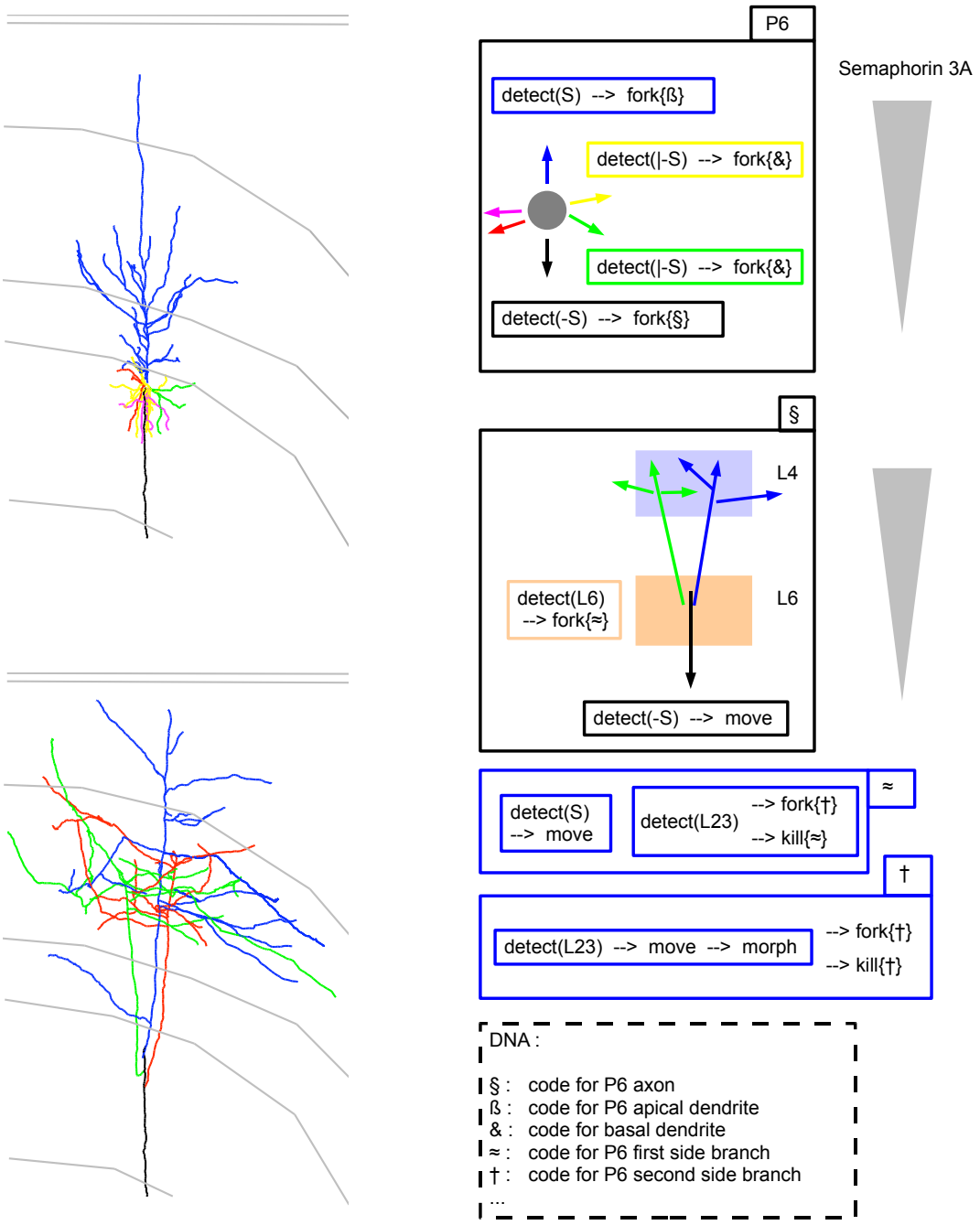


**Figure 5.5. Motives of P6 neurons**. **(A)** Neurites of a P6 neuron: the apical dendrite (blue) is the prolongation of the soma in the pial direction, and extends toward L1. The axon (black) leaves the soma in the opposite direction. The basal dendrites are extended tangentially. **(B)** The initial machine used for the neurite extension, first step in neuronal differentiation. Its correct function depends on the existence of a concentration gradient of the substance Semaphorin3A). **(C)** The main trunk of the axon goes toward the white matter, and some of the collaterals produced in L23 and L5. **(D)** The P23 axon main trunk machine. **(D)** The L23 collateral machine. **(F)** The cell's DNA contains a description of all the machines used.

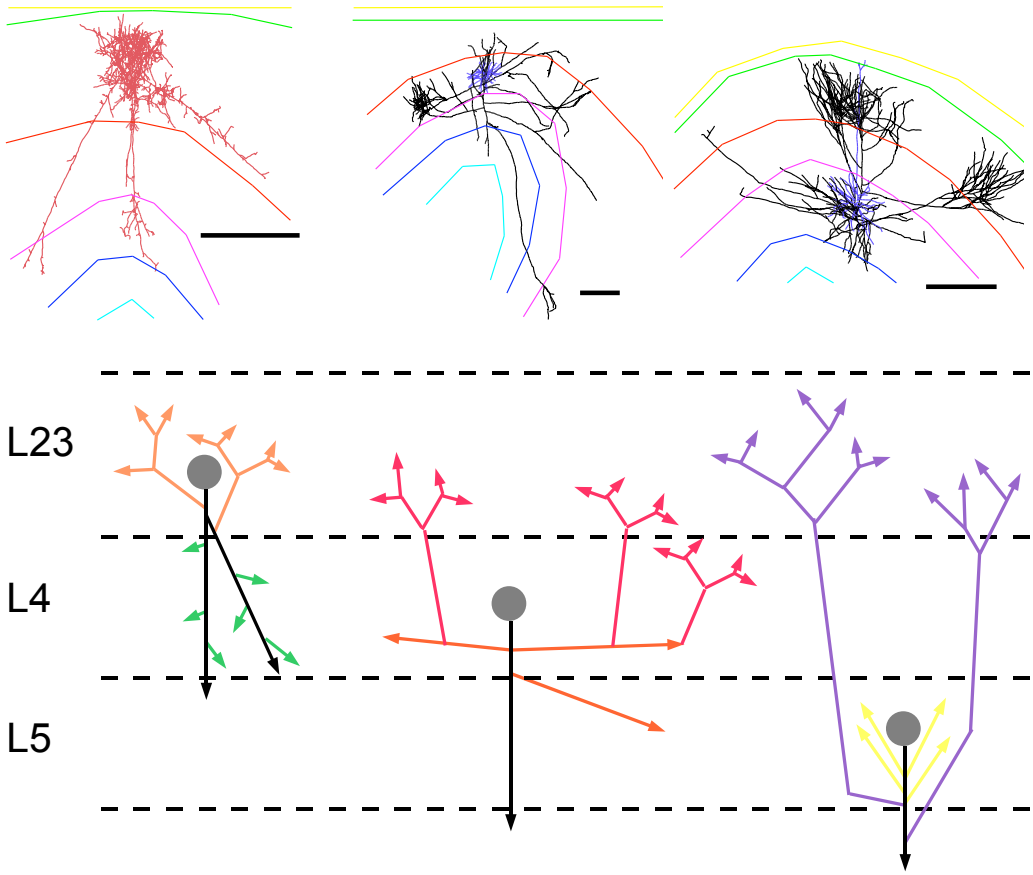


**Figure 5.6. Motives for B23, SS4, P5** (Top) Basket cells of layer 2/3, Spiny stellate cells of layer 4, Pyramidal cells of layer 4 (Scale bar: 100 $\mu m$. Neurons from cat visual cortex, reconstructed by John Anderson, INI). (Bottom) Decomposition into motives, each coded by a single machine.

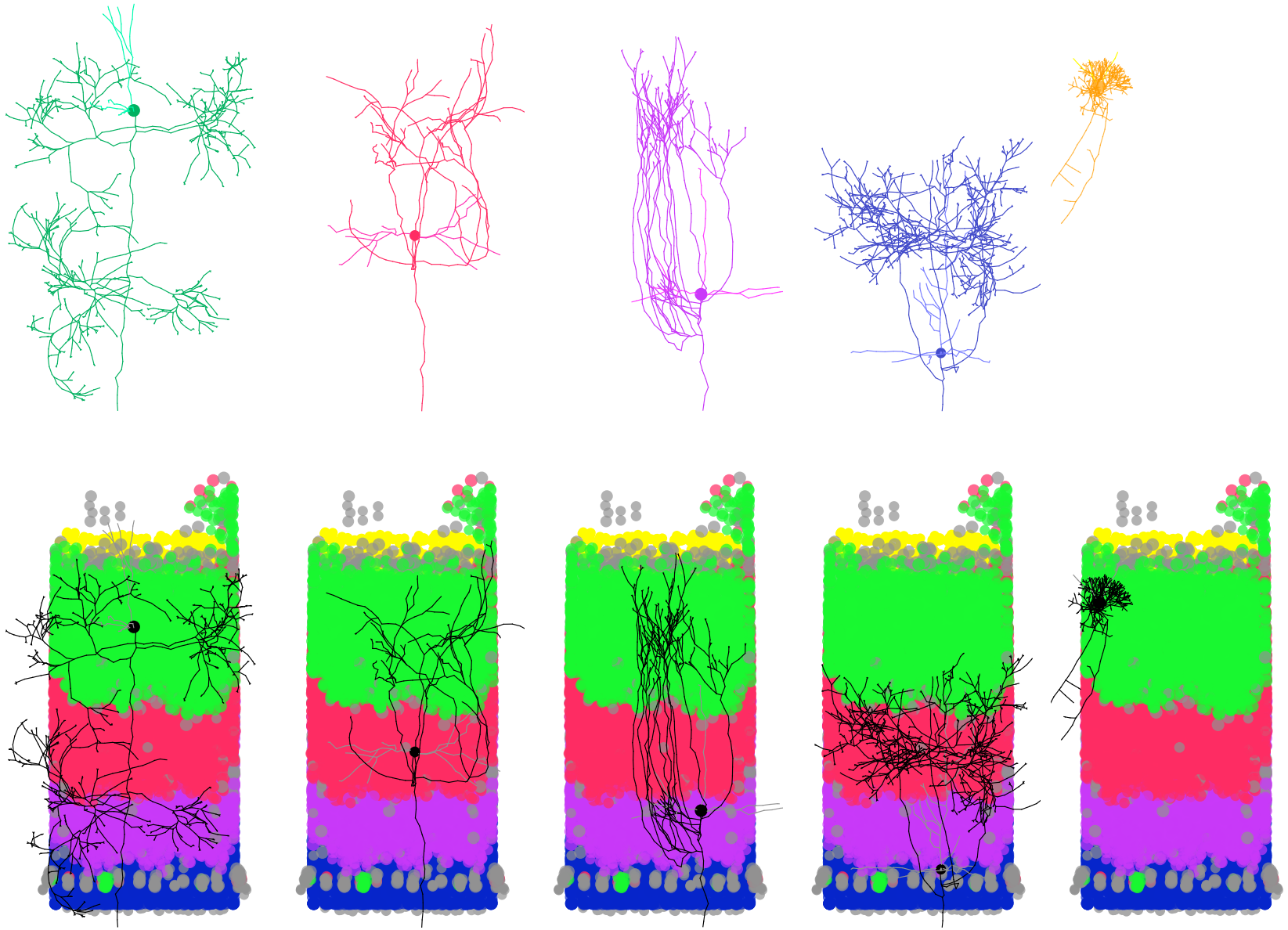

**Figure 5.7. Simulation results for five different cell types**. View in projection of the layered structure described in 5.2 (Apoptotic cells appear in grey). One cell of each layer expresses its type specific differentiating machines. An inhibitory cell that just migrated into the cortical column also differentiates. Note the layer selectivity of the projections and the border effects, for instance the bending of the branches toward the center of the column, where the signaling substances concentration is the highest. From left to right: P23, SS4, P5, P6 and B23.

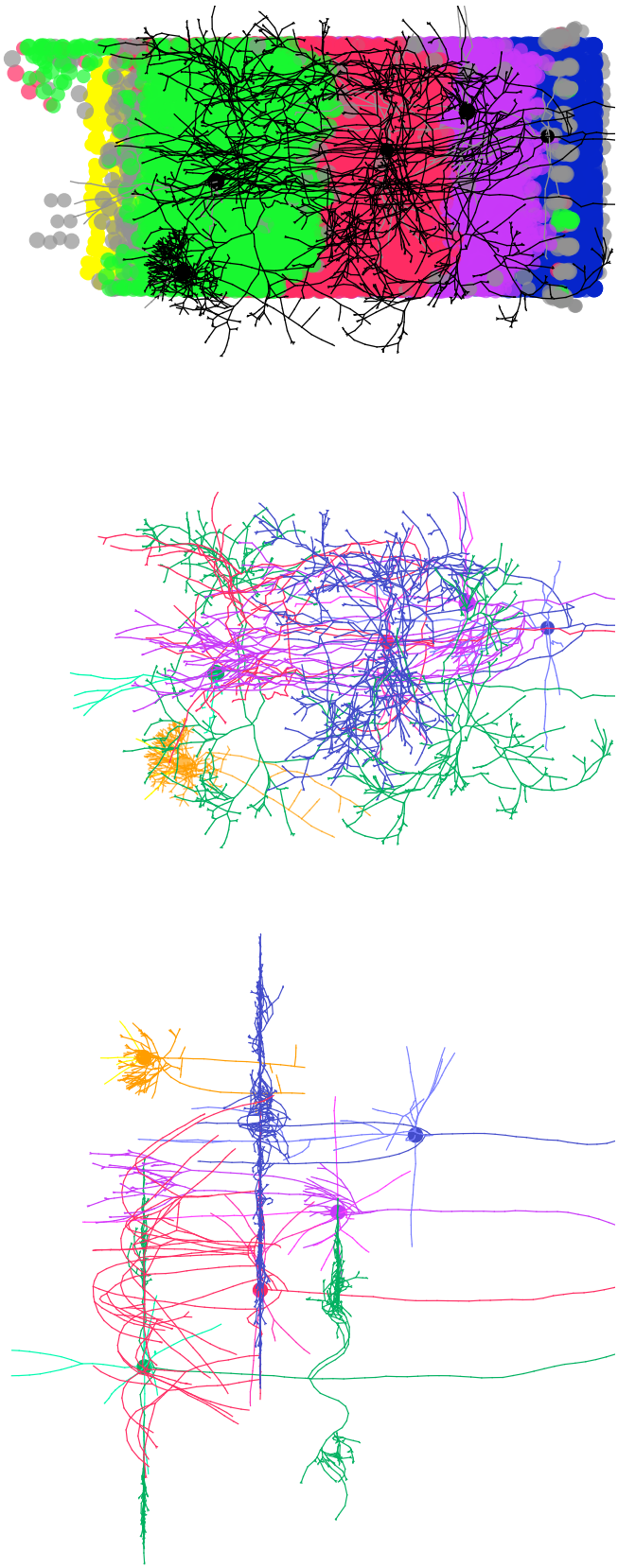

**Figure 5.8. Effect of noise and physical obstacles on the axonal and dendritic growth.** Five cells, each one with a different type-specific differentiation machine expressed are placed **(A)** in an empty environment, with perfect gradient defining the layers and **(B, C)** in the dense layered structure, where imperfectly placed cells define the chemical concentration by release of signaling molecules and represent obstacles to the growing neurites.

# Appendix A

# First version of the Instruction code

### The receptor as only instruction

This appendix section describes the 'ancestor' of the instruction code presented in Chapter 4. This old code is based on the concept of membrane receptors, the activation of which specifies the cell element's behavior. Such a receptor is composed of an intracellular domain (the action performed) and an *extracellular* segment (the signaling molecule that triggers the receptor). In the language's syntax, each domain is represented by a single character. For instance `fA` defines the code to generate a receptor elongating the neurite, moving up the gradient of a substance 'A'. The intracellular part `f` ('follow') indicates that the receptor causes the elongation of an axon up the gradient of a specific signal. The extracellular part `A` indicates that the receptor recognizes the extracellular substance 'A'. The possible actions are listed in Table A.1.

### Grammar of the code

Each branch contains a code describing the receptors that it currently expresses, but also the receptors that it might express later, and the ones that will be expressed in the future distal branches. Only a specific part of the code is executed, the rest being transported to be executed later or to be transmitted to the daughters.

The code contained in a branch has the following structure:

$$\alpha_1 \mid ... \mid \alpha_n \ [\beta] \ \{\gamma\} \ \{\delta\}.$$

| Symbol | Name | Effect |
|---|---|---|
| `f` | follow | move up a gradient |
| `a` | avoid | move down a gradient |
| `r` | retract | retract if conc. $> \theta$ |
| `b` | bifurcate | bifurcate if conc. $> \theta$ |
| `s` | branch | side-branch if conc. $> \theta$ |
| `x` | secrete | secrete an extracellular chemical |
| `k` | reset | remove all receptors and execute the next segment of code |

**Table A.1. Our first instruction language**. The different types of receptors.

Each $\alpha_i$ is a set of receptors expressed simultaneously. Initially, the branch only expresses the code before the first '|'. Under certain conditions (see below), the set is removed, and the next set $\alpha_{i+1}$ of receptors is expressed. $\beta$ is the instruction string that is passed to the new branch in case of a side-branch formation. $\gamma$ and $\delta$ the instructions transmitted to the two daughter branches in case of bifurcation of the branch. To allow for recursion, we use the following convention: if one of the string $\beta$, $\gamma$, or $\delta$ equals the symbol '_', the newly formed branch inherits exactly the same code than the mother branch.

## First examples

We now present five basic examples of instructions, and the pattern that they generate when executed by an axon growing in an environment with a vertical gradient of an extracellular signal called 'A' (Figure A.1).

- `fA`: The instruction codes for a single receptor, with the intracellular part `f` ('follow') for gradient detection and the extracellular part `A` specifying the extracellular substance 'A'. As shown in Figure A.1 (leftmost branch), if this code is executed in a neurite, it drives the growth cone in a region of high concentration of 'A' .

- `fAkA`: this instruction codes for two receptors, `fA` and `kA`, that are expressed simultaneously. The first one is equivalent to the one discussed in the previous example. The second receptor deletes all currently active receptors when entering a region of high 'A' concentration, which stops the

elongation.

- `fAbA`: Two receptors, `fA` and `bA`. The first one moves up the gradient of 'A'; the second one induces a bifurcation of the growth cone with a probability proportional to the concentration of the extracellular substance 'A'. The two daughter branches don't elongate, since they don't contain any receptors.

- `fAbA{_}{_}` : with this code, two receptors are expressed in the initial branch: follow a gradient of 'A', and bifurcate according to the concentration of 'A'. The code transmitted to the daughter branches is specified in the two pairs of curly brakets. The character '_' means that in this case the two daughter branches inherit the same code that was present in the mother branch, i.e. they also express two receptors, and follow the gradient of 'A' and bifurcate. The mechanism continues recursively as long as the molecule 'A' is present, or as long as another mechanism interrupts the growth (in this example, the stopping mechanism depends on the diameter reduction at each bifurcation – see Appendix).

- `fAbA{fXbA{_}{_}}{fXbA{_}{_}}` : The initial growth cone expresses the same two receptors than in the previous example, and moves in a region with high concentration of 'A' where it bifurcates. But the daughter branches start with a different code, namely `fXbA{_}{_}`, under the influence of which they move up the gradient of substance 'B', and bifurcate if there is enough of 'A'. Since there is no 'B' in this environment, the gradient is null; The daughter branches move thus in a random direction. The next generations of branches express recursively the same two receptors. The difference with the previous example is that daughter branches are no longer aligned with the gradient of 'A'.

## Dependence on the environment

In the same way that the correct development of biological neurons depends crucially on their environment, our code relies on the presence of external cues. The same code executed in different environments will produce different branching patterns. Figure A.2 shows the results obtained in executing the same code in six different environments.

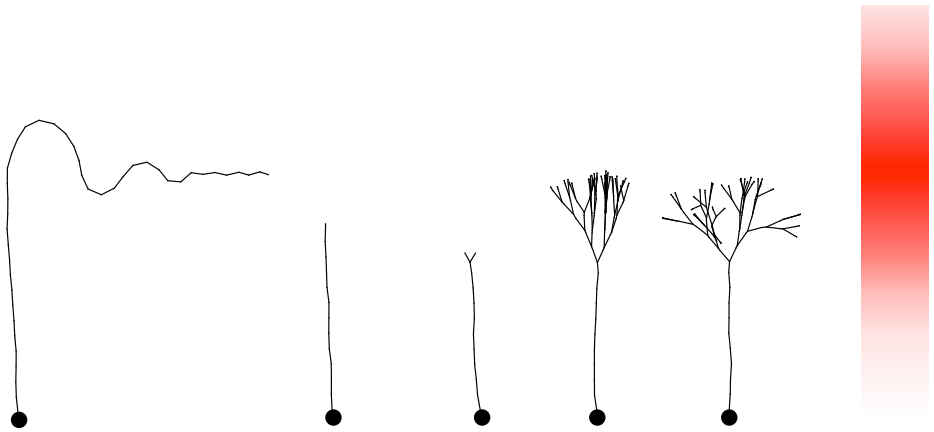

**Figure A.1.** Five different codes executed in the same environment (see text). The vertical distribution of the chemical 'A' is represented on the right, with the color intensity proportional to concentration.

The code `fMbM{fLbL{fTbL{_}{_}}{fTbL{_}{_}}}{fRbR{fTbR{_}{_}}{fTbR{_}{_}}}` describes the following procedures: move up the gradient of substance 'M', and bifurcate. The first daughter branch goes to a region with high concentration of 'R' and ramifies there. The second daughter branch goes toward a region with hight 'L' concentration, and ramifies there. Depending on the locations of the 'M','L', and 'R' signals, the neurons develop into different patterns .

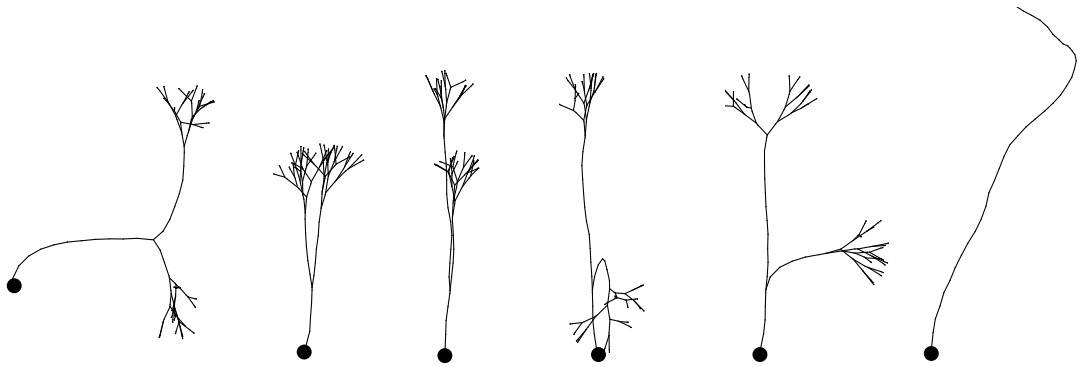

**Figure A.2.** The same instruction code executed in six different chemical environments generates six different axonal patterns.

## Sequential instructions for the formation of a simple network

Guidepost cells are (glial or neuronal) cells that participate in the formation of complex projections by serving as intermediate target. It is a computationally interesting problem, since the growth cone is first attracted toward the guidepost cells, but as soon as it reaches them, it has to move to the next target. The exact mechanism depends on the system at sake, but it implies always a chain in the receptor or the transduction chain. We address this problem by removing all active receptors, and by expressing new set of receptors which confer other properties to the growing neurite. This task is done by the receptor 'reset' (`k`).

Figure 3 shows four different populations of guidepost cells, secreting the diffusible substances 'A' (red), 'B' (violet), 'C' (blue), and 'D' (yellow). In this environment, we place test neurons, each extending a dendrite (black) and an axon (grey). The dendrites execute the following code: `fDbD{_}{_}`. The axons take a more complex path to get in the same area and ramify. The path is coded with the sequence `fAkA|fBkB|fCkC|fDkd|`. Initially only the code before the first '|' is executed, so that the axons head toward the 'A' secreting cells. Once they reach their target, the high concentration of 'A' triggers the destruction of the receptors, and implementation of the second segment of code, which directs the axon toward the 'B' secreting cells etc.

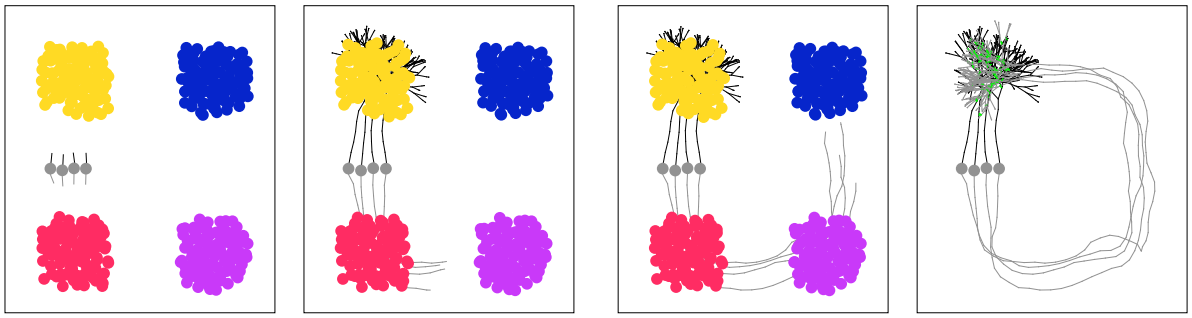

**Figure A.3.** Guidepost cells used to formed a simple circuit. Different populations of guidepost cells, secreting the diffusible substances 'A' (red), 'B' (violet), 'C' (blue), and 'D' (yellow), guide the elongation of axonal (grey) and dendritic (black) fibers, which ramify in the same area. In the rightmost figure, the guidepost cells have been removed to show the neurites.

## Discussion

The limitations of this language are obvious. We can mention for instance (1) the fact that only extracellular substance can trigger events (2) the absence of complex conditions, such as for instance [[presence

of A and of B] or presence of C] (3) the dependency on many hard coded values, such as the diameter reduction at branch points, the minimal diameter required for elongation, the concentration triggering a receptor deletion etc. These values could not be changed during run time.

An advantage of this old code was its elegant simplicity. For simple sequential instructions, it is much more concise than our new code. Figure A.3 can of course be coded with our new instruction language. But the genome, specified in an XML file as described in Section 4.6, is more than a hundred lines long!

# Chapter 6

# Classical treatment of diffusion

## 6.1 Derivation of the diffusion equation

We consider a medium in which particles move randomly without interaction (monoatomic ideal gas). Let $u(\mathbf{x}, t)$ denote the concentration of the particles, $Q_V$ the total quantity of particles inside a finite volume $V$ and $u_V$ the average concentration over the volume V. We have

$$Q_V = \int_V u(\mathbf{x}, t) d\mathbf{x}, \tag{6.1}$$

$$u_V = \frac{1}{V} \int_V u(\mathbf{x}, t) d\mathbf{x}. \tag{6.2}$$

According to Fick's first law of diffusion, the flux $\mathbf{j}(\mathbf{x}, t)$ of particles is proportional to the concentration gradient :

$$\mathbf{j}(\mathbf{x}, t) = -D(\mathbf{x}, t) \nabla u(\mathbf{x}, t) \tag{6.3}$$

where $D(\mathbf{x}, t)$ is the diffusion constant.

The continuity equation, on the other hand, states that the change in quantity of matter in any given volume $V$ is equal to the sum of the particles flowing in an out of the volume (i.e. the flux) and of the particles that are created or destroyed by respectively the sources or the sinks inside the volume :

$$\frac{d}{dt}\int_V u(\mathbf{x},t)d\mathbf{x} = -\int_{\partial V}\mathbf{j}(\mathbf{x},t)\cdot\mathbf{n}\cdot ds + \int_V f(\mathbf{x},t)d\mathbf{x} \tag{6.4}$$

where $\mathbf{n}$ is the exterior unitary vector normal to $\partial V$ and $f(\mathbf{x},t)$ the production or degradation of the particles. With the divergence theorem we obtain :

$$\int_V \frac{\partial}{\partial t}u(\mathbf{x},t) = \int_V \Big( -\nabla\cdot\mathbf{j} + f(\mathbf{x},t)\Big)d\mathbf{x}. \tag{6.5}$$

Since we haven't made any assumption on $V$, we have that everywhere

$$\frac{\partial u}{\partial t}(\mathbf{x},t) = -\nabla\cdot\mathbf{j}(\mathbf{x},t) + f(\mathbf{x},t). \tag{6.6}$$

If we insert (6.3) into (6.5) we get

$$\frac{\partial u}{\partial t}(\mathbf{x},t) = \nabla\Big(D(\mathbf{x},t)\nabla u(\mathbf{x},t)\Big) + f(\mathbf{x},t).$$

We will only consider the case where $D(\mathbf{x},t) = D$, i.e. the case where the diffusion constant remains indeed constant over time throughout the whole tissue. We thus get:

$$\frac{\partial u}{\partial t}(\mathbf{x},t) = D\nabla^2 u(\mathbf{x},t) + f(\mathbf{x},t), \tag{6.7}$$

## 6.2 Classical numerical technics

There are three major classical approaches for the numerical resolution of the diffusion equation, namely the finite difference method (FDM), the finite elements methods (FEM) and finite volume method (FVM). All three discretize the problem of finding an unknown function $u(x,t)$, i.e. they transform it into another problem with a finite number of values $u_i(t)$ to compute.

The FDM makes an approximation of the derivative. The FEM uses a functional basis onto which the unknown function $u$ is projected, and in which a variational form of the PDE is solved. The FVM divides the region of interest in small non-overlapping volumes, and with the divergence theorem transforms the terms with a divergence into surface integrals.

### 6.2.1 Finite differences

The key idea of the finite differences method is to approximate the spatial derivatives by a finite difference approximation, like in :

$$\frac{dg}{dx}(x) \approx \frac{g(x+h) - g(x)}{h}.$$

As an example, we try to solve the diffusion equation in 1 dimension:

$$\frac{\partial u}{\partial t} - D\frac{\partial^2 u}{\partial x^2} = f(x,t), \quad 0 < x < 1. \tag{6.8}$$

We first discretize the interval $[0,1]$, by defining $h = 1/(N+1)$ and $x_i = ih$ with $i = 0, 1, 2, ..., N+1$. We consider $u_i(t)$, an approximation of $u(x,t)$ at the point $x = x_i$ (cf Fig 6.1). Now we approximate the second order spatial derivative by applying twice a first-order difference and we re-write a discrete version of (6.8) as :

$$\frac{d}{dt}u_i(t) + \frac{D}{h^2}\big( - u_{i-1}(t) + 2u_i(t) - u_{i+1}(t)\big) = f(x,t)^1, \tag{6.9}$$

which is called the *semi-discretized* version of the PDE, and is in fact a system of $(N+1)$ ODE, that we can solve with classical methods, in order to get the approximation functions $u_i(t)$.

We see how this method can be generalized to higher dimensions. Nevertheless the system is computationally harder to solve (in 1D, (6.9) is a tridiagonal matrix, which is no longer the case in higher dimensions). And most of all, it usually requires the points to be placed on a regular grid, which forbids us to use our Delaunay vertices.

### 6.2.2 Finite elements

To illustrate this method, let's also consider the one-dimensional case. The first step is to multiply equation (6.8) by a function $v(x)$ and to integrate between 1 and 0 :

$$\int_0^1 \frac{\partial u}{\partial t}(x,t)v(x)dx - \int_0^1 D\frac{\partial^2 u}{\partial x^2}(x,t)v(x)dx = \int_0^1 f(x,t)v(x)dt$$

If we integrate by parts the second term we get :

$$\int_0^1 \frac{\partial u}{\partial t}(x,t)v(x)dx - D\frac{\partial u}{\partial x}(1,t)v(1) + D\frac{\partial u}{\partial x}(0,t)v(0) + \int_0^1 D\frac{\partial u}{\partial x}(x,t)v'(x)dx = \int_0^1 f(x,t)v(x)dt.$$

[1] If we have boundary conditions $u(0,t) = u(1,t) = 0\ \forall t$, then of course $\frac{d}{dt}u_i(t) = 0$ and $u_i = 0$ for $i = \{0, N\}$

If now we suppose $v(0) = v(1) = 0$, we get :

$$\int_0^1 \frac{\partial u}{\partial t}(x,t)v(x)dx + \int_0^1 D\frac{\partial u}{\partial x}(x,t)v'(x)dx = \int_0^1 f(x,t)v(x)dt, \tag{6.10}$$

which is the *weak*, or *variational* formulation of (6.8). It is the starting point of many numerical methods. We now define the space:

$$V = \{g : [0,1] \to \mathbb{R} \mid g \text{ continuous}, g' \text{piecewise continuous}, g(0) = g(1) = 0\}.$$

If $\phi_1, \phi_2, ..., \phi_N$ are $N$ linearly independent functions of $V$, we can define $V_h$ the sub-space of all linear combinations of the functions $\phi_i$. The *Galerkin approximation* is to find for all $t > 0$ a function $u_h(\cdot, t) \in V_h$ that verifies (6.10) for all functions $v \in V_h$. We write $u_h(\cdot, t)$ in the basis $\phi_1, \phi_2, ..., \phi_N$ of $V_h$:

$$u_h(x,t) = \sum_{i=1}^{N} u_i(t)\phi_i(x). \tag{6.11}$$

The values $u_i(t)$ are the components of $u_h(x,t)$ in the basis $\{\phi_i\}_{i=1,..,N}$. Inserting (6.11) into (6.10), and by choosing as function $v = \phi_j,\ j = 1, 2, ..., N$ we find :

$$\sum_{i=1}^{N} \dot{u}_i(t) \int_0^1 \phi_i(x)\phi_j(x)dx + \sum_{i=1}^{N} u_i(t) \int_0^1 D\phi_i'(x)\phi_j'(x)dx = \int_0^1 f(x,t)\phi_j(x)dx \tag{6.12}$$

for $j = 1, 2, ..., N$. We thus have a system of $N$ ordinary differential equations (one for each $\phi_j$). Solving it gives us the $u_i(t)$, the coefficients of $u_h$ in the the $\phi_h$ basis. This method is computationally heavy: before solving the system (6.12), one has to calculate numerically the integrals it contains.

In the FEM of degree 1, we define some nodes $x_i$ in the interval $[0, 1]$, and define $\phi_i(x_j) = \delta_{ij}$ with a linear interpolation between the vertices (cf Fig 6.1).

The method can be used in higher dimensions, after creation of a graph, with the functions $\phi_x$ defined similarly, taking the value 1 at the vertex $x_i$, 0 at all other vertices, and interpolated between theses points.

### 6.2.3 Finite volumes

This method is useful for computation of *conserved* values. The first step is to subdivide the space into smaller non-overlapping volume $V_i$, over which we average the concentration :

$$u_i(t) = \frac{1}{V_i} \int_{V_i} u(x,t)dt. \tag{6.13}$$

We then apply conservative arguments to each discrete volume, similarly as in (6.4):

$$\frac{d}{dt}u_i(t) = -\int_{\partial V_i} \vec{j} \cdot \vec{n} \cdot ds + \int_{V_i} f(x,t)dx$$

Finally we get a numerical approximation of the flux, with a finite difference scheme for instance:

$$\int_{\partial V_i} \vec{j} \cdot \vec{n} \cdot ds \;\approx\; \sum_{j}^{neighbors} D\frac{A}{d}\big(u_i(t) - u_j(t)\big). \tag{6.14}$$

We see that this method takes the reverse steps than the ones followed for the derivation of the diffusion equation. It is thus equivalent to directly apply with "fickian" diffusion between all our neighboring PhysicalNodes, which is very similar to the way we treat mechanics, computing only local interactions between neighboring objects. The use of volume integrals allows us to define a point secretion of substances at the location of the point mass of the physical spheres or cylinders defining the neurons (the function $f(x,t)$ can be define as a sum of 'Dirac functions'). Moreover this method is simple to implement, can be made stable even with large steps and comes automatically with mass conservation. Therefore, it seems to be the perfect method to use in our context.

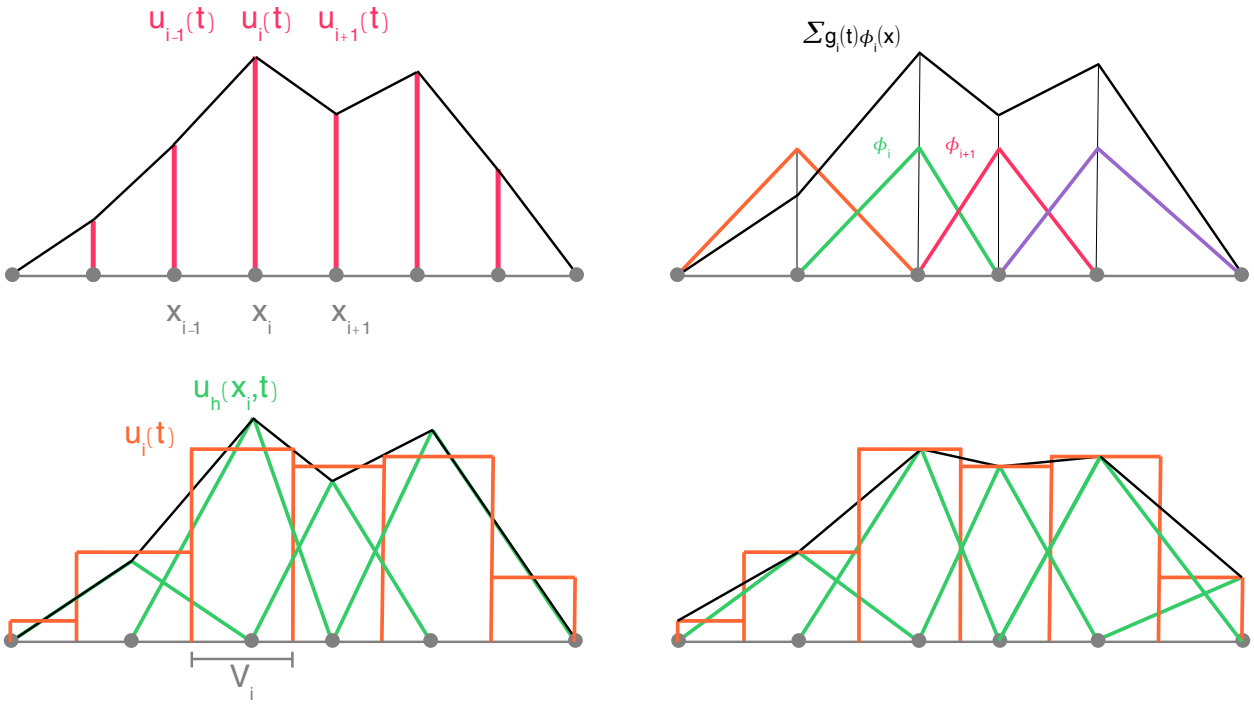


**Figure 6.1.** (A) In finite differences methods, we consider functions $u_i(t)$, which are approximations of $u(x,t)$ at position $x_i$. (B) In finite elements methods, the function $u_h(x,t)$ is the projection of the concentration profile $u(x,t)$ onto the subspace $V_h$, expressed as a linear combination of base functions $\phi_i$. (C) In finite volumes method (orange), the value $u_i(t)$ describes the average of the concentration over the domain $V_i$, and thus the exact concentration $u(x,t)$ at the node $x_i$ (the center of $V_i$) is not directly accessible. Compare with finite elements method (green), where the value of the function $u_h(x_i)$ is considered to be the exact value at the point $x_i$. (D) If we consider that the function $u_i(t)$ given by FVM is the exact concentration value at vertex $x_i$, we can use a FEM-like approach to define the concentration outside of the nodes, by linear interpolation.

# Bibliography


1. Shen Q, Wang Y, Dimos JT, Fasano CA, Phoenix TN, et al. (2006) The timing of cortical neurogenesis is encoded within lineages of individual progenitor cells. Nat Neurosci 9: 743–751.

2. Mallamaci A, Stoykova A (2006) Gene networks controlling early cerebral cortex arealization. Eur J Neurosci 23: 847–856.

3. Gaspard N, Bouschet T, Hourez R, Dimidschstein J, Naeije G, et al. (2008) An intrinsic mechanism of corticogenesis from embryonic stem cells. Nature 455: 351–357.

4. van Ooyen A (2003) Modeling Neural Development. The MIT Press.

5. Roth F (2007) Explicit design and adaptation in self-construction. Ph.D. thesis, ETH Zuerich.

6. Nagpal R (2002) Programmable self-assembly using biologically-inspired multiagent control. In: AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems. pp. 418–425.

7. Hines ML, Carnevale NT (1997) The neuron simulation environment. Neural Comput 9: 1179–1209.

8. Bower JM, Beeman D (1995) The Book of Genesis: Exploring Realistic Neural Models with the General Neural Simulation System. Electronic Library of Science, The.

9. Pecevski D, Natschlger T, Schuch K (2009) Pcsim: a parallel simulation environment for neural circuits fully integrated with python. Frontiers in Neuroinformatics 3.

10. Markram H (2006) The blue brain project. Nat Rev Neurosci 7: 153–160.

11. Alves R, Antunes F, Salvador A (2006) Tools for kinetic modeling of biochemical networks. Nat Biotechnol 24: 667–672.

12. Hilgetag CC, Barbas H (2006) Role of mechanical factors in the morphology of the primate cerebral cortex. PLoS Comput Biol 2: e22.

13. Huang H, Kamm RD, Lee RT (2004) Cell mechanics and mechanotransduction: pathways, probes, and physiology. Am J Physiol Cell Physiol 287: C1–11.

14. Shefi O, Harel A, Chklovskii DB, Ben-Jacob E, Ayali A (2004) Biophysical constraints on neuronal branching. Neurocomputing 58-60: 487-495.

15. Dennerll TJ, Lamoureux P, Buxbaum RE, Heidemann SR (1989) The cytomechanics of axonal elongation and retraction. J Cell Biol 109: 3073–3083.

16. Lamoureux P, Ruthel G, Buxbaum RE, Heidemann SR (2002) Mechanical tension can specify axonal fate in hippocampal neurons. J Cell Biol 159: 499–508.

17. Kageyama R, Ohtsuka T, Shimojo H, Imayoshi I (2008) Dynamic notch signaling in neural progenitor cells and a revised view of lateral inhibition. Nat Neurosci 11: 1247–1251.

18. Chilton JK (2006) Molecular mechanisms of axon guidance. Dev Biol 292: 13–24.

19. Abdi H (1994) A neural network primer. Journal of Biological Systems 2(3): 244-283.

20. Holland JH (1975) Adaptation in Natural and Artificial Systems. Ann Arbor, MI, USA: University of Michigan Press.

21. Goldberg DE (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional.

22. Montana D, Davi L (1989) Training feedforward neural networks using genetic algorithms. In: Proceedings of the International Joint Conference on Artificial Intelligence.

23. Xin Y (1999) Evolving artificial neural networks. In: Proceedings of the IEEE. volume 87, pp. 1423–1447.

24. Kitano H (1990) Designing neural networks using genetic algorithms with graph generation system. Complex Systems Journal 4: 461-476.

25. Boers E, Kuiper H, Happel B, Sprinkhuizen-Kuyper I (1993) Designing modular artificial neural networks. In: Proceedings of Computing Science in The Netherlands. pp. 87-96.

26. Vaario J, Shimohara K (1997) Synthesis of developmental and evolutionary modeling of adaptive autonomous agents. In: CANN '97: Proceedings of the 7th International Conference on Artificial Neural Networks. Springer-Verlag, pp. 721-726. London, UK.

27. Nolfi S (2002) Handbook of brain theory and neural networks, Second Edition, Cambridge, MA: MIT Press, chapter Evolution and Learning in neural networks. pp. 415-418.

28. Rust A, Adams R, George S, Bolouri H (1996) Artificial evolution: Modelling the evelopment of the retina. Technical report, Engineering Research and Development Centre, University of Hertfordshire.

29. Cangelosi A, Nolfi S, Parisi D (1994) Cell division and migration in a 'genotype' for neural networks. Network : Computation in neural system 5: 497-515.

30. Kitano H (1995) A simple model of neurogenesis and cell differentiation based on evolutionary large-scale chaos. Artif Life 2: 79-99.

31. Eggenberger P (2001) Axonal growth in evolutionary neurogenesis. Artif Life Robotics 5: 137-141.

32. Gruau F, Ratajszczak JY, Wiber G (1995) A neural compiler. Theor Comput Sci 141: 1-52.

33. Roth F, Siegelmann H, Douglas RJ (2007) The self-construction and -repair of a foraging organism by explicitly specified development from a single cell. Artif Life 13: 347–368.

34. Stiefel KM, Sejnowski TJ (2007) Mapping function onto neuronal morphology. J Neurophysiol 98: 513–526.

35. Lindenmayer A (1968) Mathematical models for cellular interactions in development. parts 1 and 2. J Theor Biol 18: 20–315.

36. Hamilton P (1993) A language to describe the growth of neurites. Biol Cybern 68: 559–565.

37. Mulchandani K (1995) Morphological Modelling of Neurons. Ph.D. thesis, Texas A&M University.

38. Ascoli GA, Krichmar JL, Scorcioni R, Nasuto SJ, Senft SL (2001) Computer generation and quantitative morphometric analysis of virtual neurons. Anat Embryol (Berl) 204: 283–301.

39. Kliemann W (1987) A stochastic dynamical model for the characterization of the geometrical structure of dendritic processes. Bull Math Biol 49: 135–152.

40. vanPelt J, Verwer RW (1983) The exact probabilities of branching patterns under terminal and segmental growth hypotheses. Bull Math Biol 45: 269–285.

41. Samsonovich AV, Ascoli GA (2005) Statistical determinants of dendritic morphology in hippocampal pyramidal neurons: A hidden markov model. Hippocampus 15: 166–183.

42. da Fontoura Costa L, Coelho RC (2005) Growth-driven percolations: the dynamics of connectivity in neuronal systems. The European Physical Journal B - Condensed Matter and Complex Systems Volume 47: 571-581.

43. Luczak A (2006) Spatial embedding of neuronal trees modeled by diffusive growth. J Neurosci Methods 157: 132–141.

44. Ryder EF, Bullard L, Hone J, Olmstead J, Ward MO (1999) Graphical simulation of early development of the cerebral cortex. Comput Methods Programs Biomed 59: 107–114.

45. Shinbrot T (2006) Simulated morphogenesis of developmental folds due to proliferative pressure. J Theor Biol 242: 764–773.

46. Samuels DC, Hentschel HG, Fine A (1996) The origin of neuronal polarization: a model of axon formation. Philos Trans R Soc Lond B Biol Sci 351: 1147–1156.

47. Cai AQ, Landman KA, Hughes BD (2006) Modelling directional guidance and motility regulation in cell migration. Bull Math Biol 68: 25–52.

48. Kiddie G, McLean D, Ooyen AV, Graham B (2005) Development, Dynamics and Pathology of Neuronal Networks: From Molecules to Functional Circuits, Progress in Brain Research 147, J Van Pelt and M. Kamermans and C.N.Levelt and A. Van Ooyen and G.J.A. Ramakers and P.R. Roelfsema, chapter Biologically plausible models of neurite outgrowth. pp. 67-80.

49. Maskery S, Shinbrot T (2005) Deterministic and stochastic elements of axonal guidance. Annu Rev Biomed Eng 7: 187–221.

50. Goodhill GJ, Gu M, Urbach JS (2004) Predicting axonal response to molecular gradients with a computational model of filopodial dynamics. Neural Comput 16: 2221–2243.

51. Krottje JK, van Ooyen A (2007) A mathematical framework for modeling axon guidance. Bull Math Biol 69: 3–31.

52. Hentschel HG, van Ooyen A (1999) Models of axon guidance and bundling during development. Proc Biol Sci 266: 2231–2238.

53. van Ooyen A, Willshaw DJ (1999) Competition for neurotrophic factor in the development of nerve connections. Proc Biol Sci 266: 883–892.

54. Stepanyants A, Hirsch JA, Martinez LM, Kisvrday ZF, Ferecsk AS, et al. (2008) Local potential connectivity in cat primary visual cortex. Cereb Cortex 18: 13–28.

55. Reber M, Burrola P, Lemke G (2004) A relative signalling model for the formation of a topographic neural map. Nature 431: 847–853.

56. de Gennes PG (2007) Collective neuronal growth and self organization of axons. Proc Natl Acad Sci U S A 104: 4904–4906.

57. Jessell TM (2000) Neuronal specification in the spinal cord: inductive signals and transcriptional codes. Nat Rev Genet 1: 20–29.

58. Ingham PW, McMahon AP (2001) Hedgehog signaling in animal development: paradigms and principles. Genes Dev 15: 3059–3087.

59. Kobayashi D, Kobayashi M, Matsumoto K, Ogura T, Nakafuku M, et al. (2002) Early subdivisions in the neural plate define distinct competence for inductive signals. Development 129: 83–93.

60. Angevine JB, Bodian D, Coulombre AJ, Jr MVE, Hamburger V, et al. (1970) Embryonic vertebrate central nervous system: revised terminology. The Anatomical Record 166(2): 257-61.

61. Bystron I, Blakemore C, Rakic P (2008) Development of the human cerebral cortex: Boulder committee revisited. Nat Rev Neurosci 9: 110–122.

62. Goetz M, Huttner WB (2005) The cell biology of neurogenesis. Nat Rev Mol Cell Biol 6: 777–788.

63. Rakic P (1971) Guidance of neurons migrating to the fetal monkey neocortex. Brain Res 33: 471–476.

64. Gal JS, Morozov YM, Ayoub AE, Chatterjee M, Rakic P, et al. (2006) Molecular and morphological heterogeneity of neural precursors in the mouse neocortical proliferative zones. J Neurosci 26: 1045–1056.

65. Noctor SC, Martnez-Cerdeo V, Ivic L, Kriegstein AR (2004) Cortical neurons arise in symmetric and asymmetric division zones and migrate through specific phases. Nat Neurosci 7: 136–144.

66. Dehay C, Kennedy H (2007) Cell-cycle control and cortical development. Nat Rev Neurosci 8: 438–450.

67. Takiguchi-Hayashi K, Sekiguchi M, Ashigaki S, Takamatsu M, Hasegawa H, et al. (2004) Generation of reelin-positive marginal zone cells from the caudomedial wall of telencephalic vesicles. J Neurosci 24: 2286–2295.

68. Bayer SA, Altman J (1990) Development of layer 1 and the subplate in the rat neocortex. Exp Neurol 107: 48–62.

69. Rakic P (1972) Mode of cell migration to the superficial layers of fetal monkey neocortex. J Comp Neurol 145: 61–83.

70. Jones EG (2009) The origins of cortical interneurons: mouse versus monkey and human. Cereb Cortex 19: 1953–1956.

71. Nadarajah B, Parnavelas JG (2002) Modes of neuronal migration in the developing cerebral cortex. Nat Rev Neurosci 3: 423–432.

72. Hatanaka Y, Hisanaga SI, Heizmann CW, Murakami F (2004) Distinct migratory behavior of early- and late-born neurons derived from the cortical ventricular zone. J Comp Neurol 479: 1–14.

73. Encyclopedia of Computational Mechanics.

74. D'Arcangelo G, Miao GG, Chen SC, Soares HD, Morgan JI, et al. (1995) A protein related to extracellular matrix proteins deleted in the mouse mutant reeler. Nature 374: 719–723.

75. Hack I, Bancila M, Loulier K, Carroll P, Cremer H (2002) Reelin is a detachment signal in tangential chain-migration during postnatal neurogenesis. Nat Neurosci 5: 939–945.

76. Foerster E, Jossin Y, Zhao S, Chai X, Frotscher M, et al. (2006) Recent progress in understanding the role of reelin in radial neuronal migration, with specific emphasis on the dentate gyrus. Eur J Neurosci 23: 901–909.

77. McConnell SK, Kaznowski CE (1991) Cell cycle dependence of laminar determination in developing neocortex. Science 254: 282–285.

78. Frantz GD, McConnell SK (1996) Restriction of late cerebral cortical progenitors to an upper-layer fate. Neuron 17: 55–61.

79. Hatanaka Y, Murakami F (2002) In vitro analysis of the origin, migratory behavior, and maturation of cortical pyramidal cells. J Comp Neurol 454: 1–14.

80. Mortimer D, Fothergill T, Pujic Z, Richards LJ, Goodhill GJ (2008) Growth cone chemotaxis. Trends Neurosci 31: 90–98.

81. Kalil K, Szebenyi G, Dent EW (2000) Common mechanisms underlying growth cone guidance and axon branching. J Neurobiol 44: 145–158.

82. Portera-Cailliau C, Weimer RM, Paola VD, Caroni P, Svoboda K (2005) Diverse modes of axon elaboration in the developing neocortex. PLoS Biol 3: e272.

83. Sorribas H, Padeste C, Tiefenauer L (2002) Photolithographic generation of protein micropatterns for neuron culture applications. Biomaterials 23: 893–900.

84. Yu TW, Bargmann CI (2001) Dynamic regulation of axon guidance. Nat Neurosci 4 Suppl: 1169–1176.

85. Dickson BJ (2002) Molecular mechanisms of axon guidance. Science 298: 1959–1964.

86. Song H, Poo M (2001) The cell biology of neuronal navigation. Nat Cell Biol 3: E81–E88.

87. Yoshikawa S, Thomas JB (2004) Secreted cell signaling molecules in axon guidance. Curr Opin Neurobiol 14: 45–50.

88. Buck KB, Zheng JQ (2002) Growth cone turning induced by direct local modification of microtubule dynamics. J Neurosci 22: 9358–9367.

89. Ziv NE, Garner CC (2004) Cellular and molecular mechanisms of presynaptic assembly. Nat Rev Neurosci 5: 385–399.

90. Schaller G, Meyer-Hermann M (2004) Kinetic and dynamic delaunay tetrahedralizations in three dimensions. Computer Physics Communications 162: 9.

91. Barth T, Ohlberger M (2004) Encyclopedia of Computational Mechanics, Volume 1, Fundamentals, John Wiley & Sons, chapter Finite volume methods: foundation and analysis. pp. 439–474.

92. Ng HN, Grimsdale RL (1996) Computer graphics techniques for modeling cloth. IEEE Computer Graphics and Applications 16: 28–41.

93. Ward K, Bertails F, Kim TY, Marschner SR, Cani MP, et al. (2007) A survey on hair modeling: Styling, simulation, and rendering. IEEE Transactions on Visualization and Computer Graphics 13: 213–234.

94. Schaller G, Meyer-Hermann M (2005) Multicellular tumor spheroid in an off-lattice voronoi-delaunay cell model. Phys Rev E Stat Nonlin Soft Matter Phys 71: 051910.

95. Pattana S (2006) Division d'un milieu cellulaire sous contraintes mecaniques. Utilisation de la mecanique des materiaux granulaires. Ph.D. thesis, Universit Montpellier II.

96. Dennerll TJ, Joshi HC, Steel VL, Buxbaum RE, Heidemann SR (1988) Tension and compression in the cytoskeleton of pc-12 neurites. ii: Quantitative measurements. J Cell Biol 107: 665–674.

97. Lamoureux P, Buxbaum RE, Heidemann SR (1989) Direct evidence that growth cones pull. Nature 340: 159–162.

98. Aeschlimann M (2000) Biophysical models of axonal pathfinding. Ph.D. thesis, University of Lausanne.

99. van Ooyen A, Graham BP, Ramakers GJA (2001) Competition for tubulin between growing neurites during development. Neurocomputing 38-40: 73-78.

100. Kiddie G, McLean D, Ooyen AV, Graham B (2005) Biologically plausible models of neurite outgrowth. Prog Brain Res 147: 67–80.

101. Castellani V, Bolz J (1997) Membrane-associated molecules regulate the formation of layer-specific cortical circuits. Proc Natl Acad Sci U S A 94: 7030–7035.

102. Dantzker JL, Callaway EM (1998) The development of local, layer-specific visual cortical axons in the absence of extrinsic influences and intrinsic activity. J Neurosci 18: 4145–4154.

103. Massobrio P, Massobrio G, Martinoia S (2007) Multi-program approach for simulating recorded extracellular signals generated by neurons coupled to microelectrode arrays. Neurocomputing 70: 2467–2476.

104. Stepanyants A, Hof PR, Chklovskii DB (2002) Geometry and structural plasticity of synaptic connectivity. Neuron 34: 275–288.

105. Goddard NH, Hucka M, Howell F, Cornelis H, Shankar K, et al. (2001) Towards neuroml: model description methods for collaborative modelling in neuroscience. Philos Trans R Soc Lond B Biol Sci 356: 1209–1228.

106. Collier JR, Monk NA, Maini PK, Lewis JH (1996) Pattern formation by lateral inhibition with feedback: a mathematical model of delta-notch intercellular signalling. J Theor Biol 183: 429–446.

107. Goodhill GJ, Richards LJ (1999) Retinotectal maps: molecules, models and misplaced data. Trends Neurosci 22: 529–534.

108. Willshaw D (2006) Analysis of mouse epha knockins and knockouts suggests that retinal axons programme target cells to form ordered retinotopic maps. Development 133: 2705–2717.

109. Goodhill GJ (2003) A theoretical model of axon guidance by the robo code. Neural Comput 15: 549–564.

110. Hutchins BI, Kalil K (2008) Differential outgrowth of axons and their branches is regulated by localized calcium transients. J Neurosci 28: 143–153.

111. de Lima AD, Gieseler A, Voigt T (2009) Relationship between gabaergic interneurons migration and early neocortical network activity. Dev Neurobiol 69: 105–123.

112. Turrigiano GG (2008) The self-tuning neuron: synaptic scaling of excitatory synapses. Cell 135: 422–435.

113. Butz M, Worgotter F, van Ooyen A (2009) Activity-dependent structural plasticity. Brain Res Rev .

114. Polleux F, Morrow T, Ghosh A (2000) Semaphorin 3a is a chemoattractant for cortical apical dendrites. Nature 404: 567–573.

115. Davis L, Dou P, DeWit M, Kater SB (1992) Protein synthesis within neuronal growth cones. J Neurosci 12: 4867–4877.

116. Campbell DS, Holt CE (2001) Chemotropic responses of retinal growth cones mediated by rapid local protein synthesis and degradation. Neuron 32: 1013–1026.

117. Berg JM, Tymoczko JL, Stryer L (2002) Biochemistry, Fifth Edition : International Version. W. H. Freeman.

118. Wen Z, Zheng JQ (2006) Directional guidance of nerve growth cones. Curr Opin Neurobiol 16: 52–58.

119. Kolch W (2005) Coordinating erk/mapk signalling through scaffolds and inhibitors. Nat Rev Mol Cell Biol 6: 827–837.

120. Locasale JW, Chakraborty AK (2008) Regulation of signal duration and the statistical dynamics of kinase activation by scaffold proteins. PLoS Comput Biol 4: e1000099.

121. Uemura E, Carriquiry A, Kliemann W, Goodwin J (1995) Mathematical modeling of dendritic growth in vitro. Brain Res 671: 187–194.

122. Binzegger T, Douglas RJ, Martin KAC (2005) Axons in cat visual cortex are topologically self-similar. Cereb Cortex 15: 152–165.

123. Bremaud P (1988) An introduction to probabilistic modeling. Springer-Verlag.

124. Rall W (1959) Branching dendritic trees and motoneuron membrane resistivity. Exp Neurol 1: 491–527.

125. Cuntz H, Borst A, Segev I (2007) Optimization principles of dendritic structure. Theor Biol Med Model 4: 21.

126. Wen Q, Chklovskii DB (2008) A cost-benefit analysis of neuronal morphology. J Neurophysiol 99: 2320–2328.

127. Donohue DE, Ascoli GA (2008) A comparative computer simulation of dendritic morphology. PLoS Comput Biol 4: e1000089.

128. McAllister AK (2007) Dynamic aspects of cns synapse formation. Annu Rev Neurosci 30: 425–450.

129. Schlitt T, Brazma A (2007) Current approaches to gene regulatory network modelling. BMC Bioinformatics 8 Suppl 6: S9.

130. Vohradsky J (2001) Neural network model of gene expression. FASEB J 15: 846–854.

131. Kauffman S (1969) Homeostasis and differentiation in random genetic control networks. Nature 224: 177–178.

132. Fuss H, Dubitzky W, Downes CS, Kurth MJ (2005) Mathematical models of cell cycle regulation. Brief Bioinform 6: 163–177.

133. Tyson JJ (1991) Modeling the cell division cycle: cdc2 and cyclin interactions. Proc Natl Acad Sci U S A 88: 7328–7332.

134. Grueber WB, Yang CH, Ye B, Jan YN (2005) The development of neuronal morphology in insects. Curr Biol 15: R730–R738.

135. Sanchez-Soriano N, Tear G, Whitington P, Prokop A (2007) Drosophila as a genetic and cellular model for studies on axonal growth. Neural Dev 2: 9.

136. Montell DJ (2008) Morphogenetic cell movements: diversity from modular mechanical properties. Science 322: 1502–1505.

137. Pratt SC, Sumpter DJ, Mallon EB, Franks NR (2005) An agent-based model of collective nest choice by the ant temnothorax albipennis. Animal Behaviour 70: 1023 - 1036.

138. Hauri A (2009) Development and usage of the instruction language Fcode for neural growth simulations. Master's thesis, ETH Zuerich.

139. Binzegger T (2000) Quantitative analysis of the morphology of neurons in cat visual cortex. Ph.D. thesis, Swiss Federal Institute of Technology Zurich.

140. Wolpert L, Smith J, Jessell T, Lawrence P (2007) Principles of Development. Oxford University Press, third edition.

141. Zernicka-Goetz M, Morris SA, Bruce AW (2009) Making a firm decision: multifaceted regulation of cell fate in the early mouse embryo. Nat Rev Genet 10: 467–477.

142. Pfister S, Douglas R (2008) A genetic language for the simulation of cortex development. In: Frontiers in Neuroinformatics. Conference Abstract: Neuroinformatics 2008.

143. Polleux F, Dehay C, Goffinet A, Kennedy H (2001) Pre- and post-mitotic events contribute to the progressive acquisition of area-specific connectional fate in the neocortex. Cereb Cortex 11: 1027–1039.

144. Cajal SR (1909) Histologie du systeme nerveux de l'homme et des vertebres. A. Maloine, Paris.

145. Lorente de No R (1949) Physiology of the Nervous System, Oxford University Press, London, chapter Cerebral cortex: Architecture, intracortical connections, motor projections. pp. 288-315.

146. Martin KA, Whitteridge D (1984) Form, function and intracortical projections of spiny neurones in the striate visual cortex of the cat. J Physiol 353: 463–504.

147. Bolz J, Uziel D, Mhlfriedel S, Gllmar A, Peuckert C, et al. (2004) Multiple roles of ephrins during the formation of thalamocortical projections: maps and more. J Neurobiol 59: 82–94.